Development of a Trunk Control System using AutoDevKit
An innovative foot detection feature based on Time of Flight principle

Master's Degree Thesis

Candidate: Marco Mannino
University Supervisor: Prof. Lucia Lo Bello
ST Microelectronics Tutor: Eng. Alessandro Troisi

Department of Electrical, Electronics and Computer Engineering
Faculty of Computer Engineering

UNIVERSITÀ degli STUDI di CATANIA

26/03/2021

Contents

1 Introduction **1**
- 1.1 The Trunk Control System . . . 1
- 1.2 AutoDevKit Ecosystem . . . 3

2 Hardware **5**
- 2.1 Introduction . . . 5
- 2.2 AEK-MCU-C4MLIT1 System Control Board . . . 5
- 2.3 AEK-CON-5SLOTS1 Connection Board . . . 7
- 2.4 VL53L1X Time of Flight Sensors . . . 8
- 2.5 AEK-MOT-2DC40Y1 . . . 10
- 2.6 AEK-LED-21DISM1 . . . 11
- 2.7 AEK-AUD-C1D9031 Audio Board . . . 12
- 2.8 AEK-CON-SENSOR1 and STEVAL-MKI207V1 . . . 13
- 2.9 AGUHAJSU 2.8 Inch Touch LCD Module . . . 14

3 Software **15**
- 3.1 Introduction . . . 15
- 3.2 SPC5Studio . . . 15
  - 3.2.1 PinMap Editor . . . 17
  - 3.2.2 How to generate, compile and debug the application . . . 18
- 3.3 AutoDevKit Plug-In . . . 19
  - 3.3.1 AutoDevKit Components . . . 19
  - 3.3.2 Board View. . . . 20

4 Trunk Control Subsystems **22**
- 4.1 Introduction . . . 22
- 4.2 Foot Detection . . . 23
  - 4.2.1 Hardware Configuration . . . 23
  - 4.2.2 Software Configuration . . . 24
- 4.3 Trunk Lock and Lift . . . 25
  - 4.3.1 Hardware Configuration . . . 25
  - 4.3.2 Software Configuration . . . 26

4.4 Visual Alert 27
4.4.1 Hardware Configuration 27
4.4.2 Software Configuration 28
4.5 Acoustic Alert 29
4.5.1 Hardware Configuration 29
4.5.2 Software Configuration 30
4.6 Motion Control 31
4.6.1 Hardware Configuration 31
4.6.2 Software Configuration 32
4.7 Infotainment GUI 33
4.7.1 Hardware Configuration 33
4.7.2 Software Configuration 34

5 Advanced Gesture for Foot Detection **35**
5.1 Introduction 35
5.2 The gesture recognition 35
5.3 Detecting Area 36
5.4 Single Sensor Setup 37
5.5 Time of Flight Sensor 38
5.6 Maximum Detecting Distance 38
5.7 Detecting Base: Single Sensor 42
5.8 Detecting Base: Double Sensor 44
5.9 The Gesture Recognition 45

6 Trunk Control System **48**
6.1 Introduction 48
6.2 Hardware Setup 49
6.3 State Machine 50
6.4 DEMO 51

7 Conclusion **52**

List of Figures

1.1 Trunk Control use case 2
1.2 Trunk Control System 3

2.1 Trunk Control boards 5
2.2 AEK-MCU-C4MLIT1 System Control Board 6
2.3 AEK-CON-5SLOTS1 7
2.4 VL53L1X-SATEL 8
2.5 Time-of-Flight principle 8
2.6 SPAD array 9
2.7 AEK-MOT-2DC40Y1 10
2.8 AEK-LED-21DISM1 11
2.9 AEK-AUD-C1D9031 12
2.10 AEK-CON-SENSOR1 and STEVAL-MKI207V1 13
2.11 AGUHAJSU 2.8 inch touch LCD module 14

3.1 SPC5 Studio and AutoDevKit logos 15
3.2 Project workspace 16
3.3 Project components 16
3.4 SPC5 Studio RLA drivers 17
3.5 PinMap Editor interface windows 18
3.6 Example AutoDevKit component configuration 20
3.7 Board View button 20
3.8 Board View editor window 21

4.1 Trunk Control subsystems 22
4.2 Foot Detection basic algorithm 23
4.3 ToF sensor board pinout 24
4.4 Foot Detection state machine 24
4.5 Trunk Lock and Lift connections 25
4.6 Trunk Lock state machine 26
4.7 Visual Alert board connections 27
4.8 Light Control state machine 28

4.9 Acoustic Alert board connections 29
4.10 Acoustic Alert state machines 30
4.11 Motion Control board connections 31
4.12 Wake-Up Detection signals 32
4.13 Motion Control state machine 32
4.14 Infotainment GUI board connections 33
4.15 Infotainment GUI state machine 34

5.1 Foot Detection gesture 36
5.2 Detection constraints 36
5.3 Single sensor setup 37
5.4 Maximum Detecting Distance 38
5.5 Ideal Detection. N measurements of the ground. 39
5.6 Real Detection. N measurements detect a distribution of values. 39
5.7 Total darkness measures 40
5.8 Cloudy day measures 40
5.9 Night and artificial light measures 41
5.10 Sunny day measures 41
5.11 Detecting Base 42
5.12 Cube placement 43
5.13 Test to find the Detecting Base 43
5.14 Detecting Base with two sensors 44
5.15 Two sensors hardware setup 44
5.16 Gesture on the new Detecting Base 45
5.17 Foot Detection System states 45
5.18 Accepted sequence 46

6.1 Trunk Control System 48
6.2 Panel CAD 49
6.3 Support for ToF sensors CAD. View from Top. 49
6.4 Trunk Control state machine 50
6.5 Trunk Control DEMO. 51

Chapter 1

Introduction

The aim of this thesis work is in realizing a **Trunk Control System** to automate the opening and closing of a car trunk. It shall be affordable, easy to use and reliable. Furthermore, it shall be safe, avoiding the possibility of being dangerous for people in the proximities. The opening of the trunk shall start after a foot detection, as it happens in the solutions currently in use. However, compared to the solutions already adopted by car makers, the aim is to propose an innovative one. The innovation consists in the use of the **Time of Flight Sensors** and in the realization of a foot detection algorithm **Gesture-Based**, on which the thesis focuses. The foot detection shall require few resources but, at the same time, it shall be effective.
To build the *Trunk Control System*, some automotive products made by **STMicroelectronics** are used. In particular, we adopt the **AutoDevKit** technologies, indispensable for fast prototyping of the *Trunk Control System*.
This Chapter describes what is a *Trunk Control System* and introduce the development tools. Chapters 2 and 3 contain respectively the descriptions of the Hardware and Software tools used. Chapter 4 focuses on the implementation of all the *Trunk Control* subsystems taken individually, with the exception of the gesture recognition, discussed instead in Chapter 5. In Chapter 6, the DEMO is presented to show all features of the *Trunk Control System* developed. And finally, the conclusions are drawn in Chapter 7.

1.1 The Trunk Control System

As stated before, the *Trunk Control* is a system which make it easier to access the trunk of a car. An example of use case consists in a person, with busy hands, who wants to access the car trunk in a simple way. Surely, performing a foot gesture to open the trunk is easier than having to open it manually.

Figure 1.1: *Trunk Control use case*

As it can be guessed from the previous example, the essential features required for a *Trunk Control* design are at least three: the *foot recognition*, the *trunk lock* and the *trunk lift*. We intend to add higher levels of safety to the standard system. These consists in preventing the system activation if the vehicle is in motion, in adding a visual and acoustic signal during the opening and closing phase, and, even more important, in allowing the system to stop if the trunk finds an obstacle.
Since the system to be realized includes several functions, it is convenient to break it down into many subsystems. Those identified are:

- **Foot Detection**: It is the key portion of the entire *Trunk Control System*. It detects a specific gesture normally performed with a feet to allow or deny the opening procedure.
- **Trunk Lock**: to unlock the trunk at the begin of the opening procedure and to lock it at the end of the closing procedure.
- **Trunk Lift**: to raise or lower the trunk.
- **Visual Alert**: for visual signaling.
- **Acoustic Alert**: for acoustic signaling.
- **Motion Alert**: to prevent the trunk opening if the car is in motion even in case of engine off.
- **Infotainment GUI**: to report the status of the system and for user interaction.

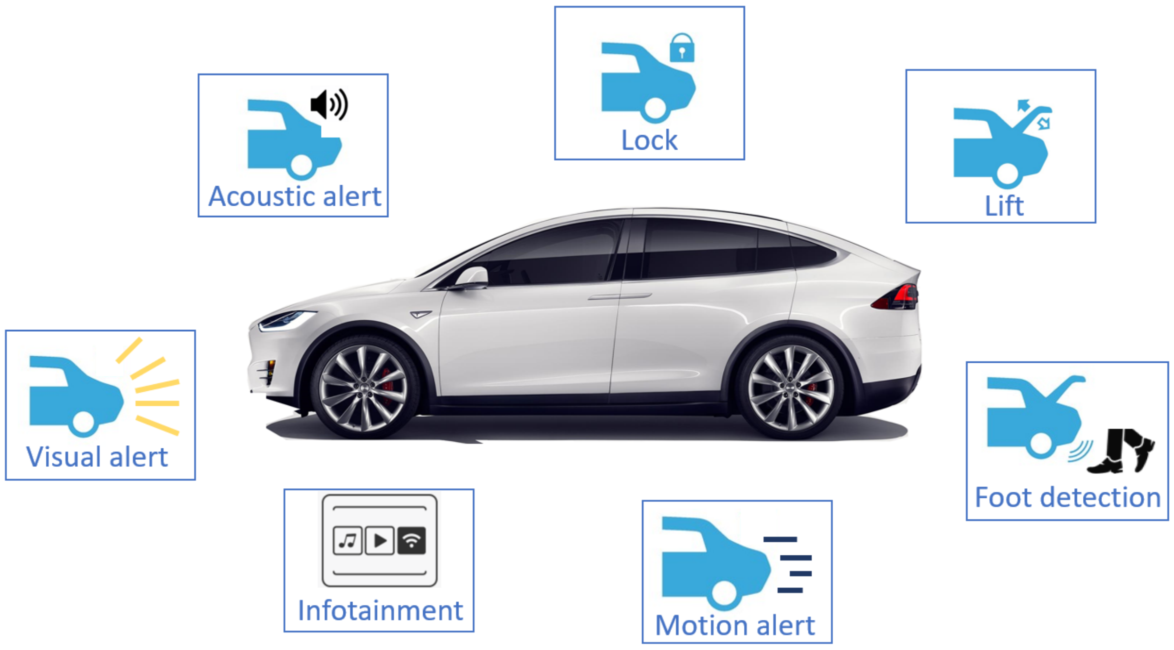


Figure 1.2: *Trunk Control System*

1.2 AutoDevKit Ecosystem

AutoDevKit is a fast prototyping ecosystem for automotive applications. It provides hardware and software tools for making the applications development easy and fast. These tools are:

- AEK-MCU Discovery boards, for evaluating specific automotive microcontroller and and AEK-xxx functional boards, implementing specific functions like lighting or motor control.
- Pre-assembled AEKD-demonstrators board implementing an automotive system solution.
- STSW-AUTODEVKIT software package, an SPC5STUDIO Eclipse plugin to facilitate firmware development for applications adopting *AutoDevKit* boards.

For each subsystem, an *AutoDevKit* functional board is used, according to Table 1.1. All subsystems are coordinated by the board AEK-MCU-C4MLIT1, hosting the

Subsystem	**Boards**
Foot Detection	VL53L1X-SATEL
Trunk Lock and Lift	AEK-2DC40Y1
Visual Alert	AEK-LED-21DISM1
Acoustic Alert	AEK-AUD-C1D9031
Motion Alert	AEK-CON-SENSOR1 and STEVAL-MKI207V1
Infotainment	GUHAJSU 2.8 Inch Touch LCD Module

Table 1.1: *Correspondence between Subsystem and Functional Board*

SPC58EC80E5 32-bit Power Architecture automotive Chorus 4MB microcontroller. The hardware used for the *Trunk Control System* is described in the next Chapter.

Chapter 2

Hardware

2.1 Introduction

As anticipated in the previous chapter, the Trunk Control System consists of seven boards handled by the AEK-MCU-C4MLIT1 and connected through the AEK-CON-5SLOTS1 connector board.

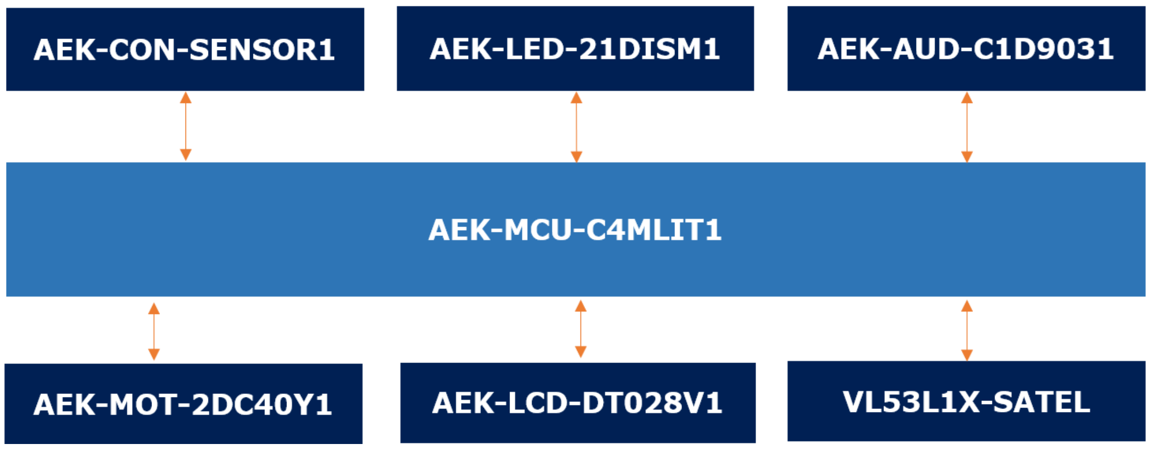


Figure 2.1: *Trunk Control boards*

2.2 AEK-MCU-C4MLIT1 System Control Board

This board features the 32-bit SPC58EC80E5 microcontroller, heart of the whole system. The board has an integrated programmer (by PLS) for MCU debugging and programming and a USB Virtual COM port for serial output to a PC running a standard terminal program.

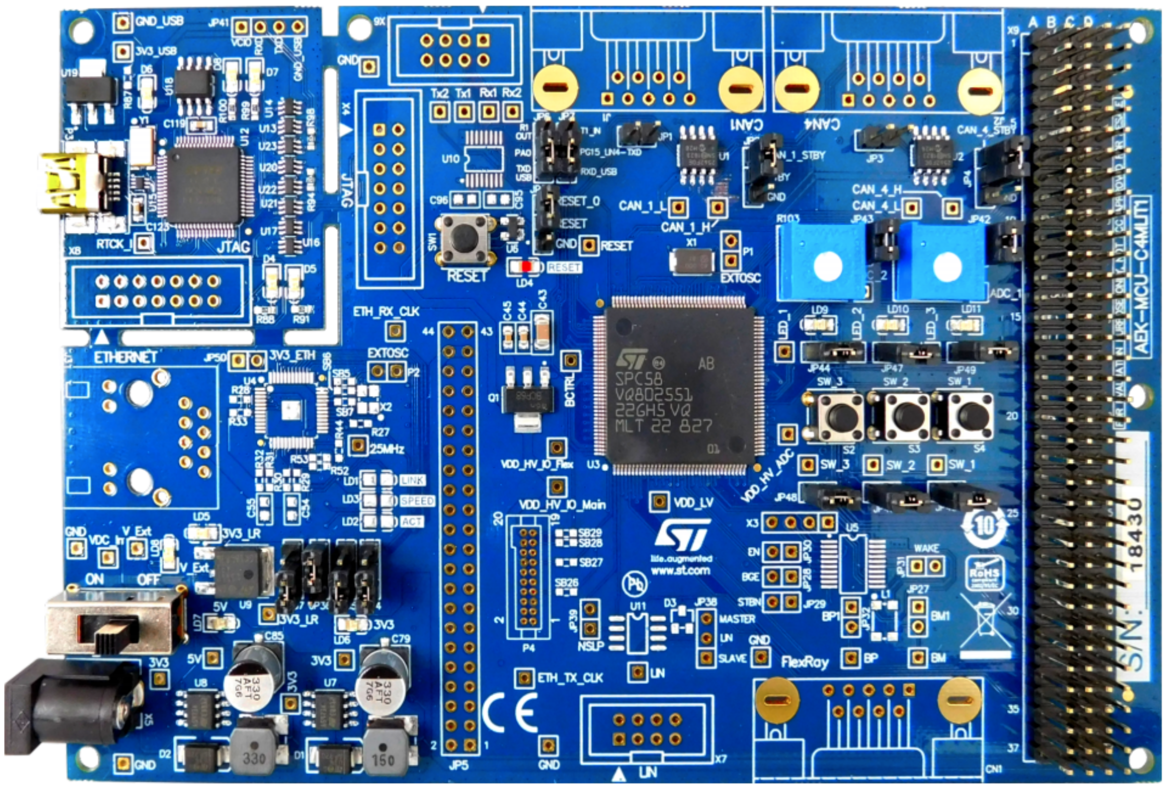


Figure 2.2: *AEK-MCU-C4MLIT1 System Control Board*

The 32-bit SPC58EC80E5 microcontroller has the following features:

- 2 main dual-issue 32-bit CPUs built on ®Power Architecture technology, operating up to 180MHz
- 4224 KB (4096 KB code Flash + 128 KB data Flash) on-chip flash memory
- 384 KB on-chip general-purpose SRAM (+ 128 KB local data RAM: 64 KB included in each CPU)
- 8 DSPIs available
- 8 (Programmable Interrupt Timer) PIT channels available
- 2 x 32 eMIOS channels available
- Up to 96 channels SARADC available
- 8 MCAN available

2.3 AEK-CON-5SLOTS1 Connection Board

The AEK-CON-5SLOTS1 connector board is designed to facilitate connections and reduce development time while prototyping automotive applications with an SPC5x Discovery board. The set of 4x37 extension connectors allows developers to reconfigure pin assignments for ADCs, Timers, GPIOs, DSPIs and other peripherals coming from SPC5 Discovery boards and generate different pin assignment patterns on the extension connectors.

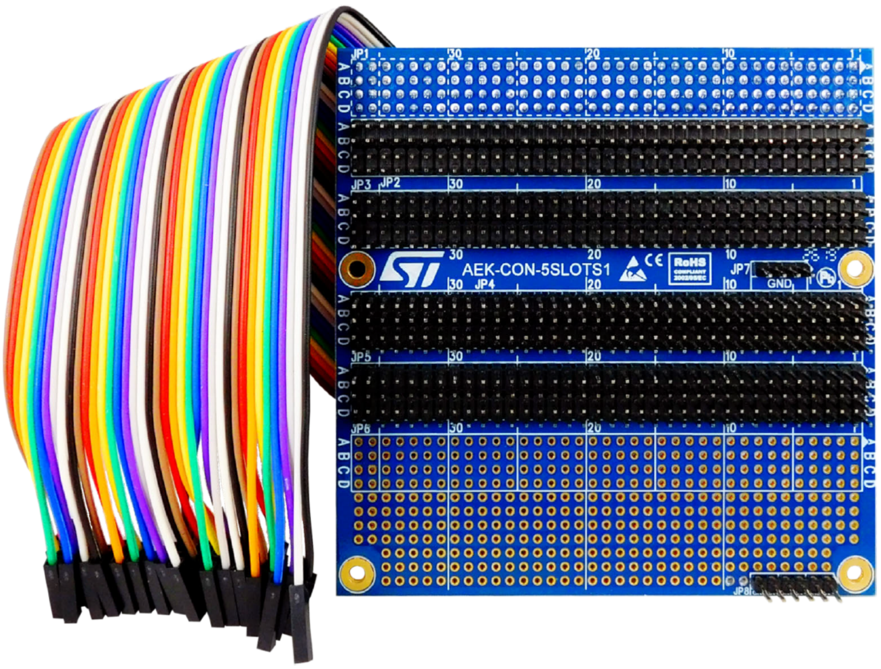


Figure 2.3: *AEK-CON-5SLOTS1*

The AEK-CON-5SLOTS1 board has the following features:

- Fast and easy extension of 4x37 connector on SPC5x discovery boards
- Single female connector to plug the board onto an SPC5x 4x37 connector
- Two male connectors with same outputs as 4x37 connector
- Two configurable male connectors 4x37 whose pins can be reassigned to connect different functional boards
- Additional ground and 5V pins
- Bare section available on the board for specific application purposes
- Female-to-female jump wire set included

2.4 VL53L1X Time of Flight Sensors

The VL53L1X device is an advanced optical device allowing distances measurements and target detection. The device incorporates the Time-of-Flight (ToF) a patent from STMicroelectronics able to measure the distance between a sensor and an object based on the time difference between the emission of a VCSEL light signal and its return to the sensor, after being reflected by the object.

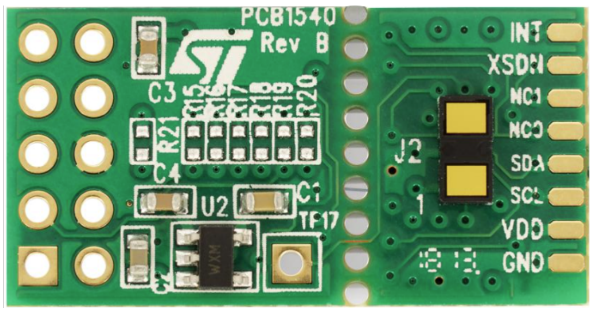


Figure 2.4: *VL53L1X-SATEL*

The VL53L1X embeds the emitter, a vertical cavity surface emitting laser (VCSEL), and the receiver, an array of single photon avalanche diodes (SPADs).
The photons emitted by the laser reflect on the target and trigger avalanches when coming back to the SPAD. The time between the emission of the photons and the avalanche (ToF) is measured and translated into a target distance in the range result register.

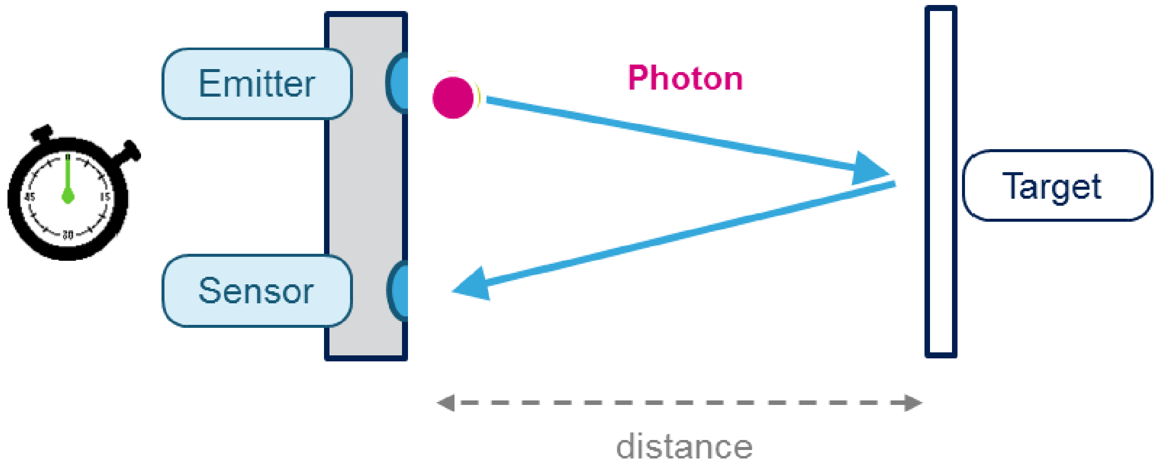


Figure 2.5: *Time-of-Flight principle*

The measured distance value is calculated using the following formula:

$$Measured\,Distance = \frac{Photon\,Travel\,Time}{2} \times Light\,Speed \tag{2.1}$$

Although the VCSEL emitted beam is not perfectly symmetrical, it can be considered as a cone. Such a beam has 97 % of the optical power present in a cone of 27°. Since no optic is used in the outgoing emission path, it is the native performance of the VCSEL. The glass present on the VCSEL is a simple IR filter to reduce interference on the t_0 time measurement. On the receiver side, there is a SPAD array covered by a convergent lens. Consequently, the system can be seen as a camera subsystem with a big granularity due to the limited number of SPADs. The full diagonal FoV of the device receiver is 27° using the 16x16 SPADs.

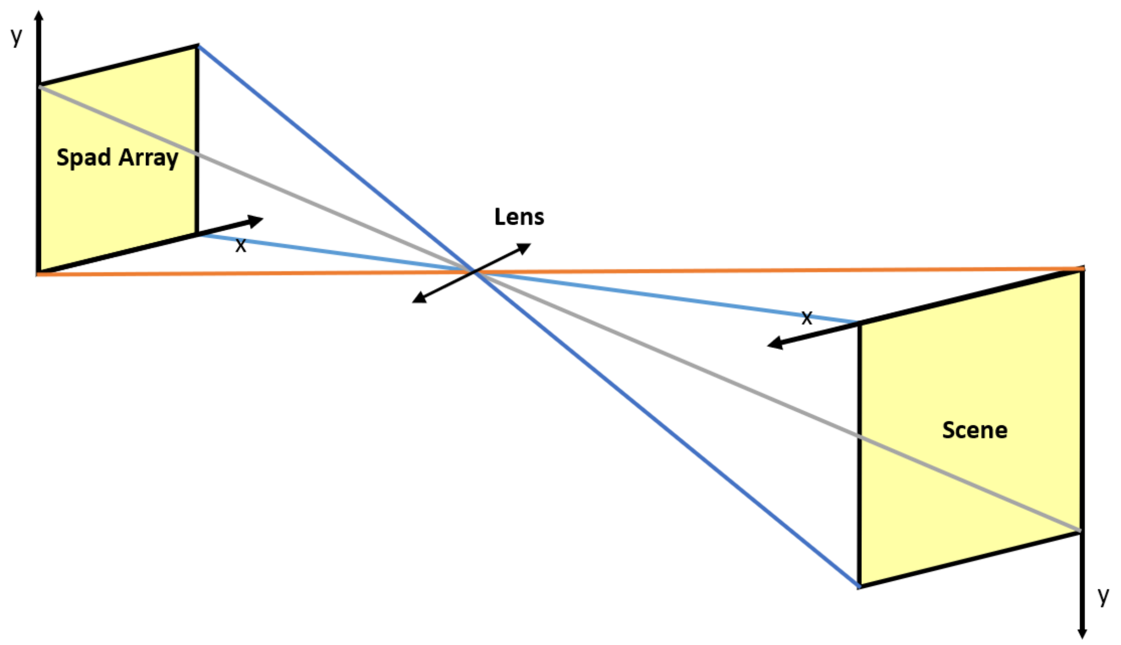


Figure 2.6: *SPAD array*

The VL53L1X sensor architecture allows ROIs (programmable Regions of Interest) with a reduced set of SPADs to be selected, thereby reducing the sensing area and allowing the receiver FoV to be reduced. The only condition is that the ROI must have an array of at least 4x4 SPADs. Summarizing, the emitter FoV cannot be modified by the VCSEL and the system keeps sending FoVs measuring 27°. Conversely, the receiver FoV can be reduced to measure only a part of the scene of interest.

2.5 AEK-MOT-2DC40Y1

The AEK-MOT-2DC40Y1 is a very compact solution for multi DC motor driving applications, embedding all the driver and signal decoding functions on the same board. It allows you to manage up to 5 DC Motors.

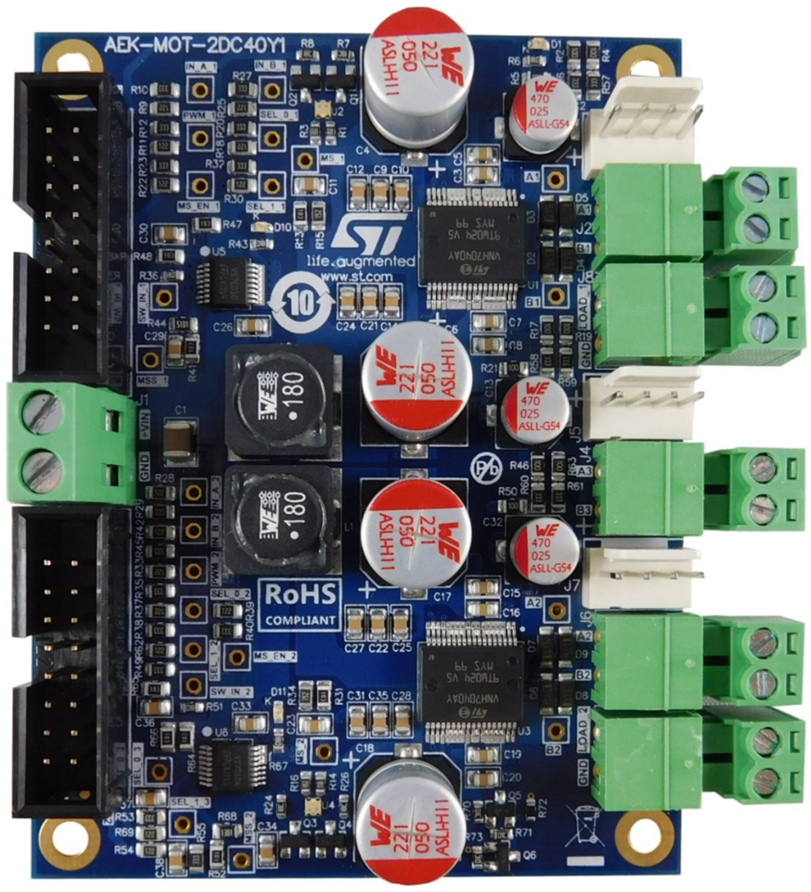


Figure 2.7: *AEK-MOT-2DC40Y1*

Each motor has a enumeration. Motor1, Motor3 and Motor5 are driven by two VNH7040AY, two automotive fully integrated H-bridge motor drivers, allowing rotation in both clockwise and counterclockwise directions. Instead Motor2 and Motor4 are driven by VN7E010AJ, two high-side drivers. All motors can rotate in parallel, except for the Motor5 because its operation depends on both the H-bringe chips. An important feature of the board consists in the current sense capability. For each motor is possible to detect how much current a motor is demanding. This feature is very useful if for example we want to detect if a motor is slowed down by some external force, since it will try to compensate requiring more current. For motors driven by the VNHs chips, the use of encoders is supported.

2.6 AEK-LED-21DISM1

The AEK-LED-21DISM1 evaluation board is designed to drive four LED strings. The board hosts two L99LD21 LED driver ICs, each of which is able to manage a single boost and two buck circuits.

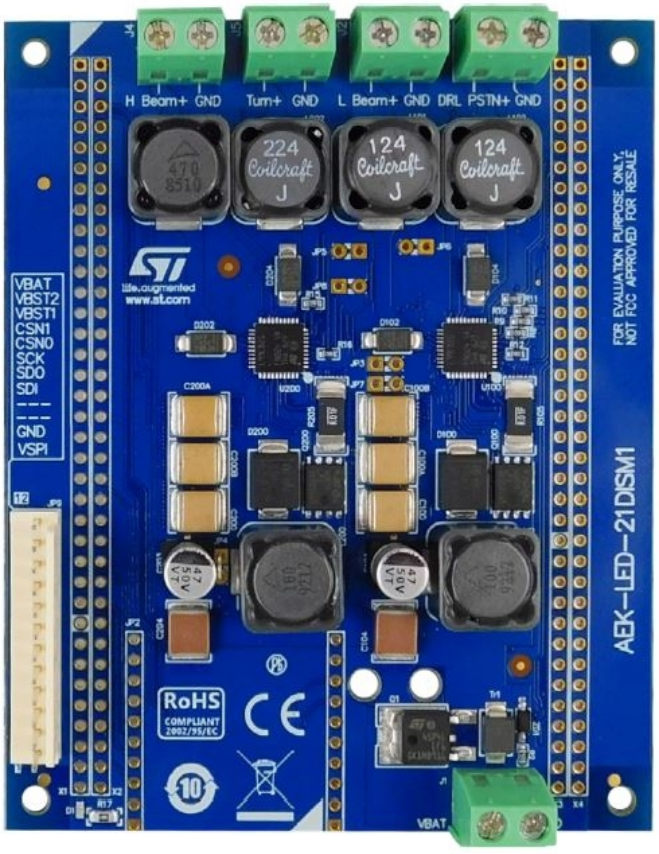


Figure 2.8: *AEK-LED-21DISM1*

The boost controller section in each L99LD21 LED driver integrates a high current gate driver for an external N-channel MOSFET and delivers a constant output voltage to two integrated buck converters. The boost controller of the two devices can be stacked to allow dual phase operation for high power applications with an interleaving pattern for improved input current ripple and current load sharing. Each L99LD21 buck converter integrates an N-channel MOSFET driven by a bootstrap circuit. The overall behavior of the board AEK-LED-21DISM1 is monitored and controlled through L99LD21 dedicated registers, which can be read or set by an external microcontroller through the fast SPI interface via a 12-pin male connector. Finally, a special Limp Home Mode implemented in the L99LD21 driver ensures that active communication with the MCU is regularly monitored.

2.7 AEK-AUD-C1D9031 Audio Board

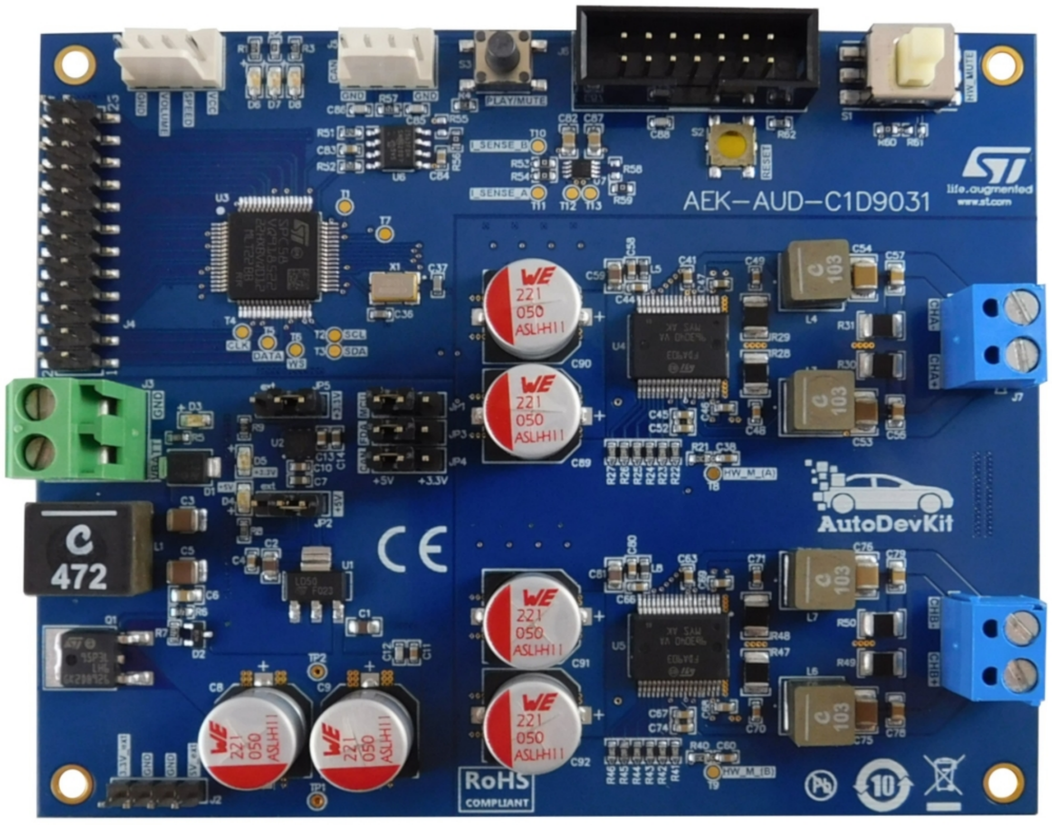


Figure 2.9: *AEK-AUD-C1D9031*

The AEK-AUD-C1D9031 is a very compact AVAS solution based on SPC582B60E1 Chorus family MCU and FDA903D Class D audio amplifiers that emits warning sounds to alert pedestrians of the presence of e-vehicles. The AEK-AUD-C1D9031 integrates two audio amplifiers in stereo mode or two separate audio channels. The board compact size allows the designer to strategically place different modules around the vehicle to ensure that warning sounds can be heard along the entire vehicle length. All the modules can be controlled by a central MCU via CAN interface. The embedded SPC582B60E1 microcontroller monitors and controls the two Class-D FDA903D power amplifiers driving the loudspeakers; the MCU sends the audio samples via I2S bus and programs the amplifiers via I2C interface.

2.8 AEK-CON-SENSOR1 and STEVAL-MKI207V1

Micro-Electro-Mechanical System (MEMS) sensors are increasingly used in automotive applications because of the advantages they can bring in safety, pollution reduction and navigation. The latest generation features very high accuracy, small size, and affordable cost.
AEK-CON-SENSOR1 is an evaluation board, dedicated to the connection of a MEMS sensor in DIL-24 socket and SPC5 microcontrollers. It is compatible all DIL-24 sensors both industrial and automotive grade. From software point view, specific drivers and application examples have been prepared to exploit the sensor functionalities.

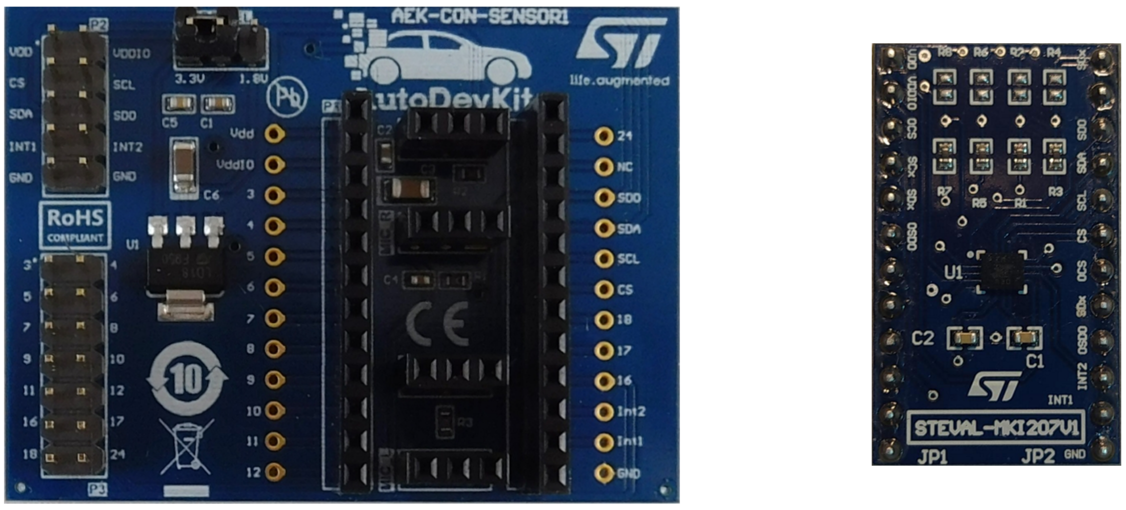


Figure 2.10: *AEK-CON-SENSOR1 and STEVAL-MKI207V1*

The STEVAL-MKI207V1 is a little evaluation board that hosts the ASM330LHH MEMS gyroscope. This board can be plugged into the AEK-CON-SENSOR1 seen before for easy evaluation of the gyroscope itself.
The ASM330LHH is a 3D digital accelerometer and 3D digital gyroscope system-in-package with a digital I^{2}C/SPI serial interface standard output, performing at 1.3 mA in combo High-Performance mode. The accelerometer features smart sleep-towake-up (Activity) and return-to-sleep (Inactivity) functions that allow advanced power saving. The device has a dynamic user-selectable full-scale acceleration range of ±2/±4/±8/±16 g and an angular rate range from ±125 to ±4000 dps. The ASM330LHH can be configured to generate interrupt signals by using hardware recognition of free-fall events, 6D orientation, activity or inactivity, and wake-up events.

2.9 AGUHAJSU 2.8 Inch Touch LCD Module

The AGUHAJSU 2.8 Inch Touch LCD Module is a board containing both the LCD and the Touch chip. Both communicates using SPI interface and are powered at 5V. The Display has a resolution of 320x240 pixels RGB and is backlit.

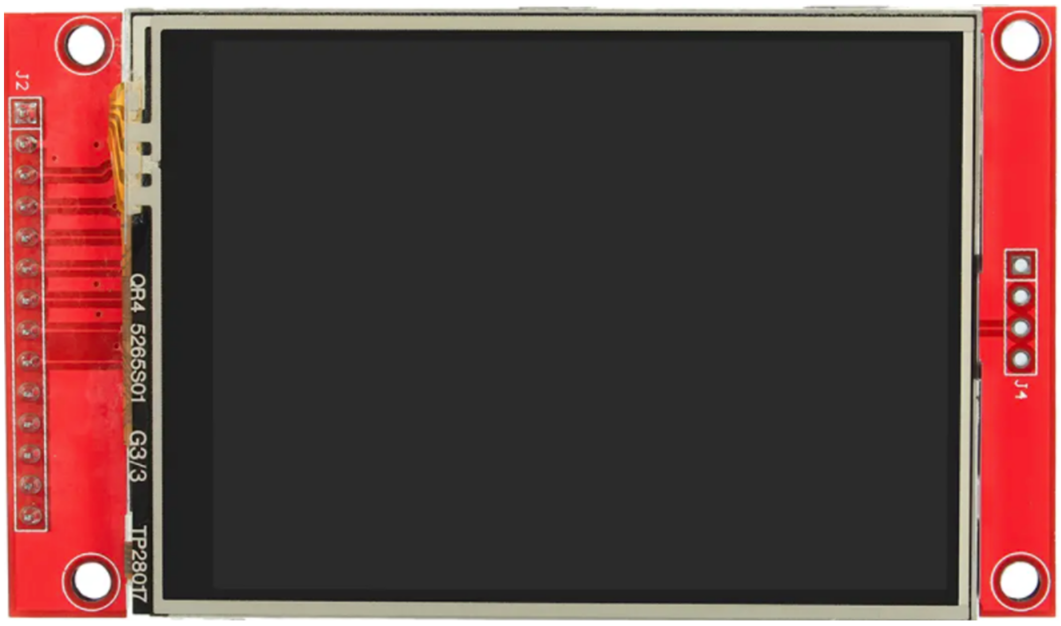


Figure 2.11: *AGUHAJSU 2.8 inch touch LCD module*

Although the display is not part of the AutoDevKit hardware ecosystem, the AutoDevKit software package provides a component which allows its use. Both chips communicate at the same SPI frequencies so just one channel can be allocated.

Chapter 3

Software

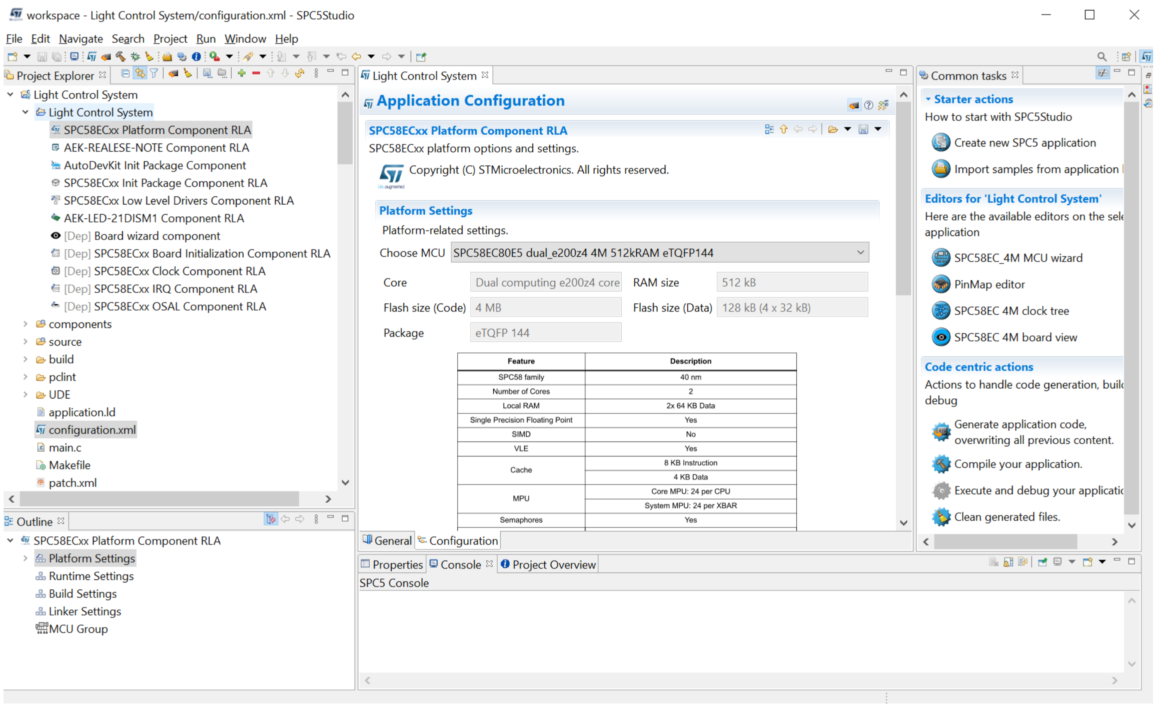
Figure 3.1: *SPC5 Studio and AutoDevKit logos*

3.1 Introduction

This chapter describes the software tools used to build the Trunk Control System: SPC5Studio and AutoDevKit software packages. The first tool enables the programming of SPC5 microcontroller families. The second one is an extension of the first enabling system-level automotive application fast prototyping.

3.2 SPC5Studio

SPC5-STUDIO is an integrated development environment (IDE) based on Eclipse. It contains a standard workspace and an extensible plug-in system environment for customizations.. The aim of SPC5-STUDIO is to maximize developer productivity of embedded applications based on SPC5 Power Architecture 32-bit microcontrollers with a single tool for evaluation, development, design and production. SPC5-STUDIO includes an application wizard to simplify project creation and configuration; it automatically solves component dependencies and generates support files.


workspace - Light Control System/configuration.xml - SPC5Studio

Project Explorer

- Light Control System
  - Light Control System
    - SPC58ECxx Platform Component RLA
    - AEK-REALESE-NOTE Component RLA
    - AutoDevKit Init Package Component
    - SPC58ECxx Init Package Component RLA
    - SPC58ECxx Low Level Drivers Component RLA
    - AEK-LED-21DISM1 Component RLA
    - [Dep] Board wizard component
    - [Dep] SPC58ECxx Board Initialization Component RLA
    - [Dep] SPC58ECxx Clock Component RLA
    - [Dep] SPC58ECxx IRQ Component RLA
    - [Dep] SPC58ECxx OSAL Component RLA
  - components
  - source
  - build
  - pclint
  - UDE
  - application.ld
  - configuration.xml
  - main.c
  - Makefile
  - patch.xml

Outline

- SPC58ECxx Platform Component RLA
  - Platform Settings
  - Runtime Settings
  - Build Settings
  - Linker Settings
  - MCU Group

Light Control System

Application Configuration

SPC58ECxx Platform Component RLA

SPC58ECxx platform options and settings.

Copyright (C) STMicroelectronics. All rights reserved.

Platform Settings

Platform-related settings.

Choose MCU: SPC58EC80E5 dual_e200z4 4M 512kRAM eTQFP144

Core: Dual computing e200z4 core — RAM size: 512 kB

Flash size (Code): 4 MB — Flash size (Data): 128 kB (4 x 32 kB)

Package: eTQFP 144

Feature	Description
SPC58 family	40 nm
Number of Cores	2
Local RAM	2x 64 KB Data
Single Precision Floating Point	Yes
SIMD	No
VLE	Yes
Cache	8 KB Instruction
	4 KB Data
MPU	Core MPU: 24 per CPU
	System MPU: 24 per XBAR
Semaphores	Yes

SPC5 Console

Common tasks

Starter actions

How to start with SPC5Studio

- Create new SPC5 application
- Import samples from application l

Editors for 'Light Control System'

Here are the available editors on the sele application

- SPC58EC_4M MCU wizard
- PinMap editor
- SPC58EC 4M clock tree
- SPC58EC 4M board view

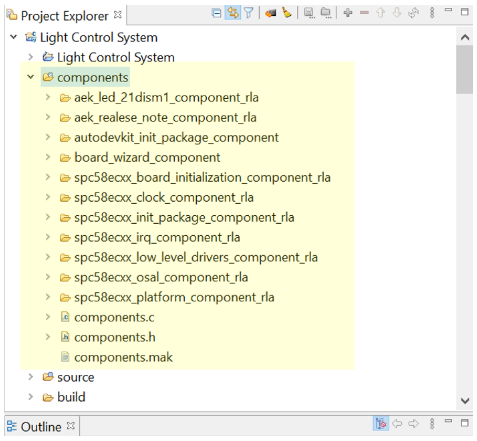
Code centric actions

Actions to handle code generation, build debug

- Generate application code, overwriting all previous content.
- Compile your application.
- Execute and debug your applicatio
- Clean generated files.

Figure 3.2: *Project workspace*

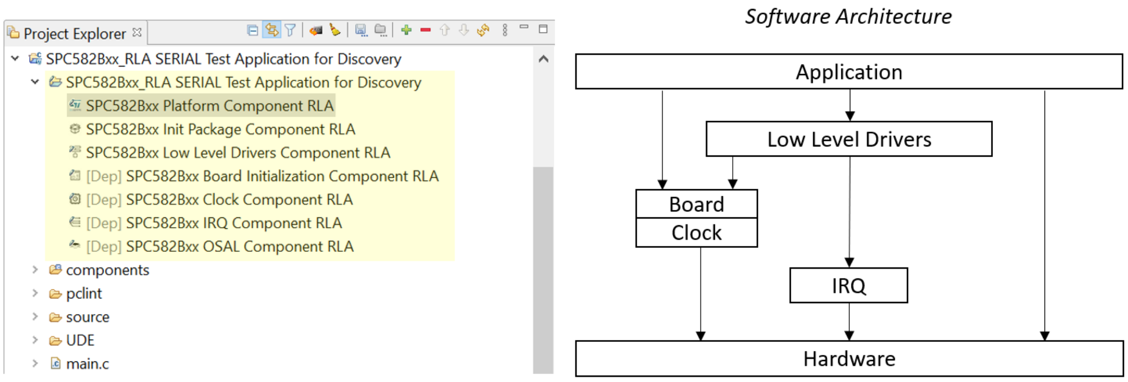
The application wizard integrates the initial components into the project, with the key elements employed by SPC5-STUDIO to generate the final application source code. Typical of layered architecture, the services in one component are provided to other components. Furthermore, the configuration of each component is supported by an intuitive GUI.


Project Explorer

- Light Control System
  - Light Control System
  - components
    - aek_led_21dism1_component_rla
    - aek_realese_note_component_rla
    - autodevkit_init_package_component
    - board_wizard_component
    - spc58ecxx_board_initialization_component_rla
    - spc58ecxx_clock_component_rla
    - spc58ecxx_init_package_component_rla
    - spc58ecxx_irq_component_rla
    - spc58ecxx_low_level_drivers_component_rla
    - spc58ecxx_osal_component_rla
    - spc58ecxx_platform_component_rla
    - components.c
    - components.h
    - components.mak
  - source
  - build

Figure 3.3: *Project components*

Register Level Access (RLA) components are low level drivers with direct access to the MCU and its peripherals such as CAN, Ethernet, DSPI, ADC, PIT and GTM. The RLA components can be added and configured through a time-saving GUI.



Project Explorer
- SPC582Bxx_RLA SERIAL Test Application for Discovery
  - SPC582Bxx_RLA SERIAL Test Application for Discovery
    - SPC582Bxx Platform Component RLA
    - SPC582Bxx Init Package Component RLA
    - SPC582Bxx Low Level Drivers Component RLA
    - [Dep] SPC582Bxx Board Initialization Component RLA
    - [Dep] SPC582Bxx Clock Component RLA
    - [Dep] SPC582Bxx IRQ Component RLA
    - [Dep] SPC582Bxx OSAL Component RLA
  - components
  - pclint
  - source
  - UDE
  - main.c

Figure 3.4: *SPC5 Studio RLA drivers*

The FreeRTOS open source real time operating system is available on request as a separate component that is compatible with the rest of the environment. SPC5-STUDIO also contains straightforward software examples for each peripheral in the MCU, which developers can use to become familiar with the specific code involved. Other advantages of SPC5-STUDIO include:

- the ability to integrate other software products from the Eclipse standard market place
- the availability of a free license GCC GNU C Compiler component
- the possibility to support industry-standard compilers
- support of multi-core microcontrollers
- a PinMap editor to facilitate MCU pin connections

3.2.1 PinMap Editor

Pins in SPC5 microcontrollers are identified as Ports because they can have different functions depending on the configuration settings in certain MCU registers:

- Peripheral related pins such as CAN, ADC, eMIOS.
- System function pins such as RESET (cannot be modified).
- Special function pins such as supply voltage pins.

The PinMap editor supports pin functionality selection by providing the possible configurations for each selected pin.

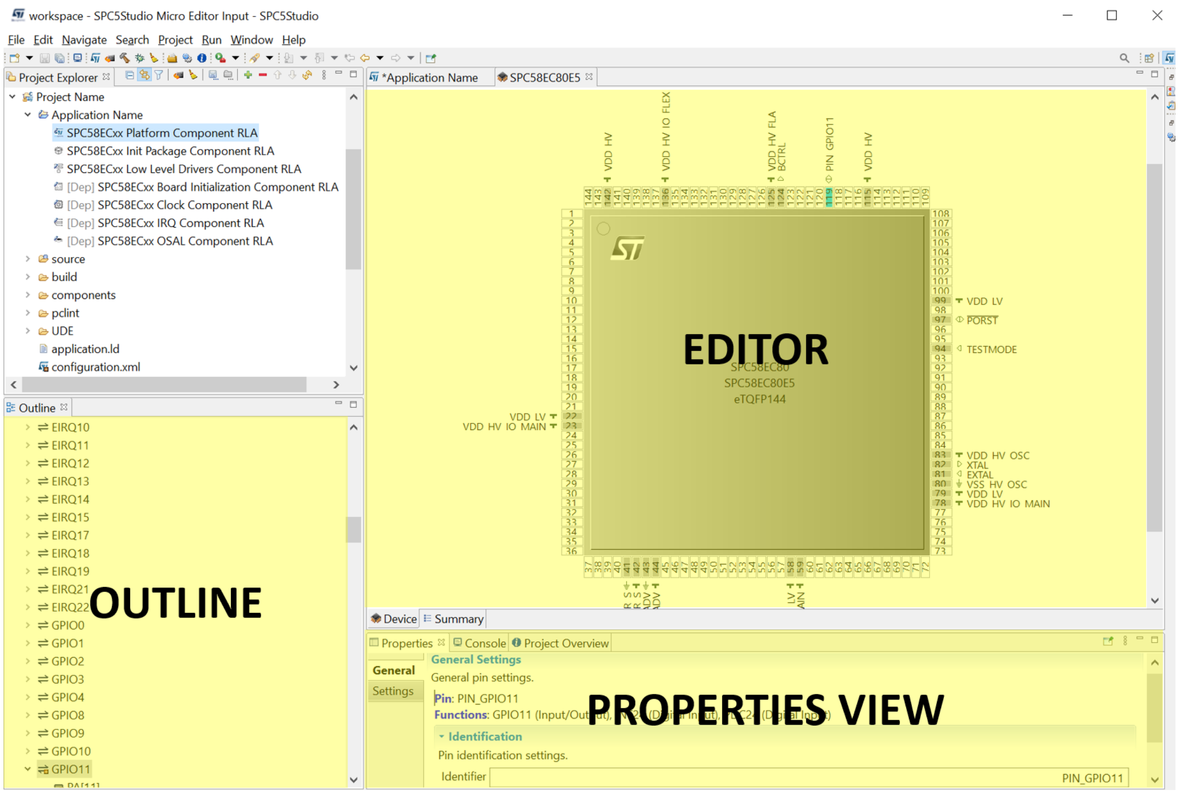


Figure 3.5: *PinMap Editor interface windows*

The Editor view is the largest window, where the pin configuration can be performed with the help of a graphical representation of the selected MCU.
The Outline view is presented in a Device → Function → Pin hierarchy, which lists all the eligible pins for a particular peripheral function.
The property view provides further information for a selected pin, including name, pin state and mode settings.

3.2.2 How to generate, compile and debug the application

SPC5Studio is a tool based on code generation. After using a component and saving the configuration the generation button must be pressed. This generates the C code that can be used in the application project. This code can be compiled using the Compile button. This will create a *debug* file inside the UDE folder that can be used to debug the application with UDE tool.

3.3 AutoDevKit Plug-In

The AutoDevKit plug-in for Eclipse extends SPC5-STUDIO into a straightforward, low-cost, time saving tool designed to help automotive application engineers evaluate, prototype, develop and deploy complex embedded systems. The main advantages of the AutoDevKit initiative are:

- Fast prototyping: with integrated hardware and software components, component compatibility checking and MCU and peripheral configuration tools.
- Flexibility: allowing the creation of new system solutions from existing solutions by adding dedicated boards for further functionality or removing unused boards.
- Hardware abstraction: if, for example, you swap your MCU between an SPC58EC Chorus 4M, an SPC584B Chorus 2M, or an SPC582B Chorus 1M, the tool will automatically re-generate pin allocations accordingly.
- High-level application APIs: AutoDevKit provides a set of function APIs able to control specific functional boards. The APIs contain functions able to exploit the chosen hardware functionality and facilitate communication with the MCU.

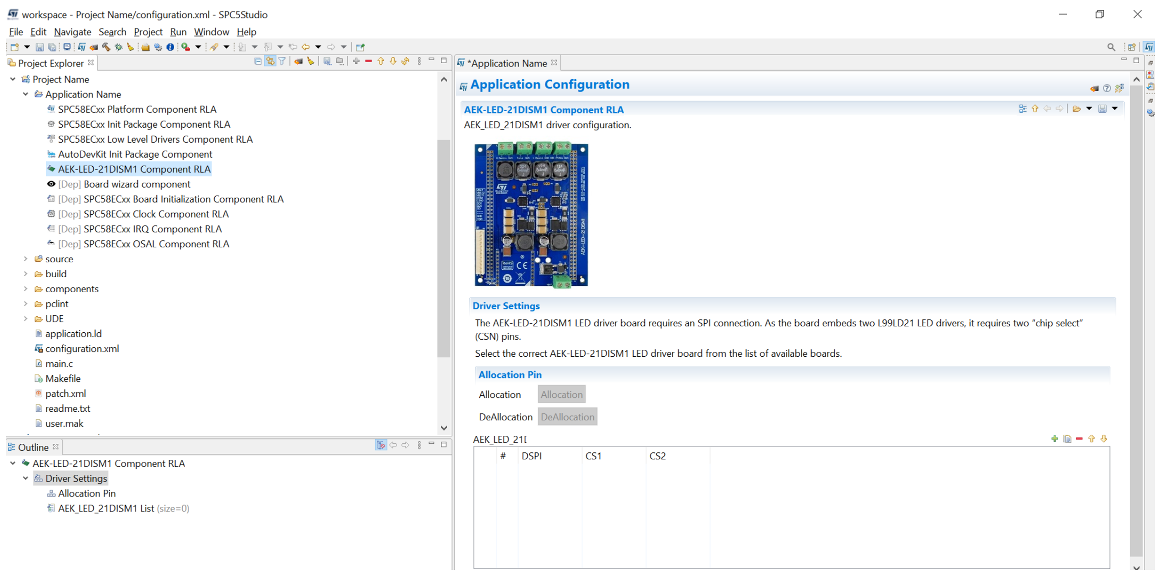
AutoDevkit provides a GUI to facilitate the configuration and setup for each supported component, together with a sample code demonstrating typical usage and applications. Another key feature is the Board View visual tool, which shows the electrical wiring between microcontroller and board connectors and the connection between the MCU board and other functional boards.

3.3.1 AutoDevKit Components

AutoDevKit adds new items to the list of the components that can be used in a SPC5-STUDIO project. AutoDevKit components are added in the same way as other components are added to an SPC5 Studio project. Each functional board component only appears in the list of the MCUs able to support it, and there is a Release Note component for all MCU platforms, which provides a summary of the components installed with the AutoDevKit plugin. The configuration of an AutoDevKit component is very similar to configuring other components in SPC5-STUDIO, the difference being two new features in the Configuration View for Pin Allocation:

- Allocation
- Deallocation

These new features automatically allocate or deallocate available MCU pins to allow the use of the MCU peripherals by newly added components. To enable these two features,the newly added component must be configured firstly.



workspace - Project Name/configuration.xml - SPC5Studio

File Edit Navigate Search Project Run Window Help

Project Explorer

- Project Name
  - Application Name
    - SPC58ECxx Platform Component RLA
    - SPC58ECxx Init Package Component RLA
    - SPC58ECxx Low Level Drivers Component RLA
    - AutoDevKit Init Package Component
    - AEK-LED-21DISM1 Component RLA
    - [Dep] Board wizard component
    - [Dep] SPC58ECxx Board Initialization Component RLA
    - [Dep] SPC58ECxx Clock Component RLA
    - [Dep] SPC58ECxx IRQ Component RLA
    - [Dep] SPC58ECxx OSAL Component RLA
  - source
  - build
  - components
  - pclint
  - UDE
  - application.ld
  - configuration.xml
  - main.c
  - Makefile
  - patch.xml
  - readme.txt
  - user.mak

Outline

- AEK-LED-21DISM1 Component RLA
  - Driver Settings
    - Allocation Pin
    - AEK_LED_21DISM1 List (size=0)

*Application Name

Application Configuration

AEK-LED-21DISM1 Component RLA

AEK_LED_21DISM1 driver configuration.

Driver Settings

The AEK-LED-21DISM1 LED driver board requires an SPI connection. As the board embeds two L99LD21 LED drivers, it requires two "chip select" (CSN) pins.

Select the correct AEK-LED-21DISM1 LED driver board from the list of available boards.

Allocation Pin

Allocation	Allocation
DeAllocation	DeAllocation

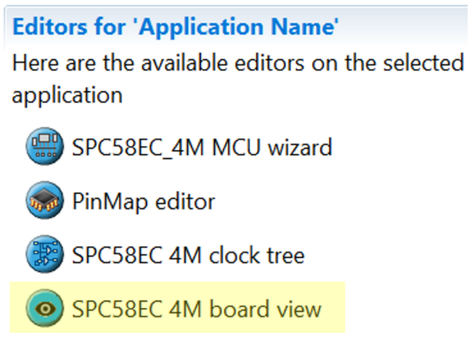
AEK_LED_21I

#	DSPI	CS1	CS2

Figure 3.6: *Example AutoDevKit component configuration*

3.3.2 Board View.

The Board View tab introduced with the AutoDevKit plugin for SPC5-STUDIO shows how MCU pins are mapped on the available board connectors. The view also provides dedicated tables to represent how the MCU allocated pins are connected to functional boards added from the AutoDevKit component library. Each table shows the mapping between functional board connectors and pins and corresponding MCU board connectors and pins. This information can render the wiring phase less time-consuming and error-prone. AutoDevKit Init Package Component must be added to the project to render the Board View button visible and to see other AutoDevKit components (functional boards).



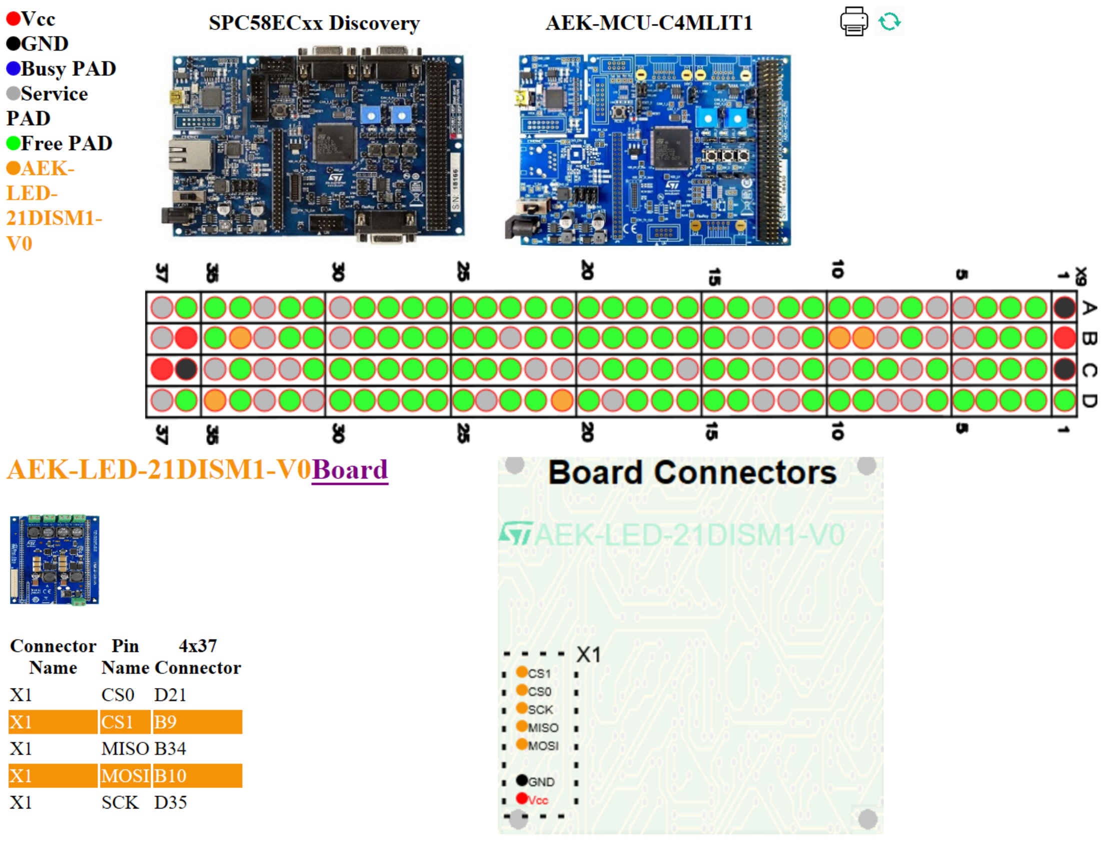
Editors for 'Application Name'

Here are the available editors on the selected application

- SPC58EC_4M MCU wizard
- PinMap editor
- SPC58EC 4M clock tree
- SPC58EC 4M board view

Figure 3.7: *Board View button*

In the example below, the colored dots in the 4x37 connector image help identify the pins required to wire the functional board to the MCU board. Some pins are required for the AEK-LED-21DISM1 board instance V0, while others are unused or reserved.



Connector Name	Pin Name	4x37 Connector
X1	CS0	D21
X1	CS1	B9
X1	MISO	B34
X1	MOSI	B10
X1	SCK	D35

Figure 3.8: *Board View editor window*

Chapter 4

Trunk Control Subsystems

4.1 Introduction

This Chapter describes the development of all the *Trunk Control Subsystems*: *Foot Detection*, *Trunk Lock*, *Trunk Lift*, *Trunk Opening*, *Motion Control*, *Visual Alert*, *Acoustic Alert*, *Infotainment GUI*. For each subsystem both hardware and software configurations are provided.

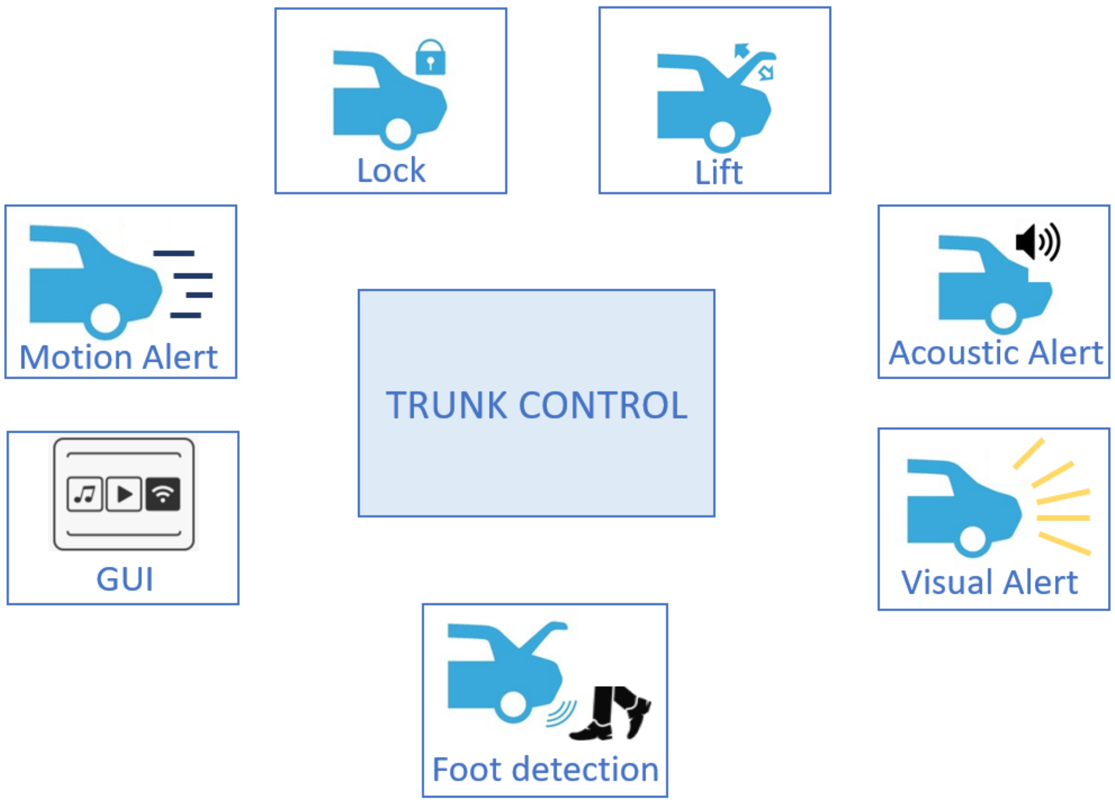


Figure 4.1: *Trunk Control subsystems*

4.2 Foot Detection

The *Foot Detection* is the main feature of the entire *Trunk Control System*. Here we describe just a simple implementation, important to understand how the ToF sensor works, while the final one will be entirely covered in Chapter 5. In this case, just one sensor is used to build this subsystem. The foot is detected if it is located at a distance less than a predetermined threshold, otherwise it is not detected.

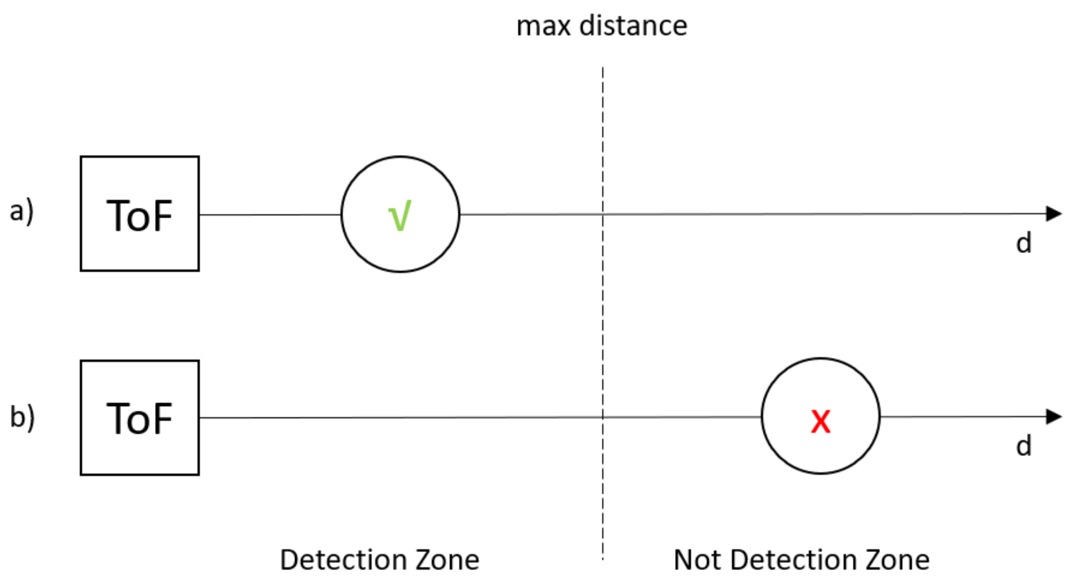


Figure 4.2: *Foot Detection basic algorithm*

4.2.1 Hardware Configuration

To use the VL53L1X sensor, at least five pins are required, as it can be seen from Figure 4.3. The first two pins, starting from the bottom, are dedicated to the GND and 3.3V power supply. The Following two pins are dedicated to the I2C Communication, carrying the Clock and the Data signals. Finally, the XSDN pin is necessary to manage the switching on and off of the sensor.

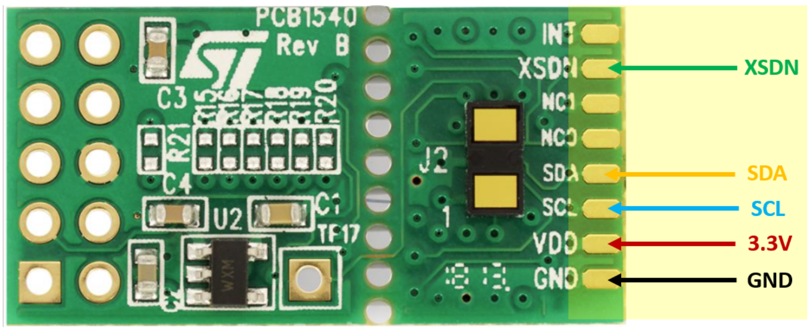


Figure 4.3: *ToF sensor board pinout*

4.2.2 Software Configuration

The software developed implements the state machine of Figure 4.4. In IDLE conditions, the system waits until a foot is detected. The microcontroller polls the sensor and receives a distance sample. If this sample is less then the selected threshold, according to Figure 1.1, the foot is detected and the chosen led turns on for a predefined WAIT_TIME. Then it turns off and the state machine returns in IDLE state.

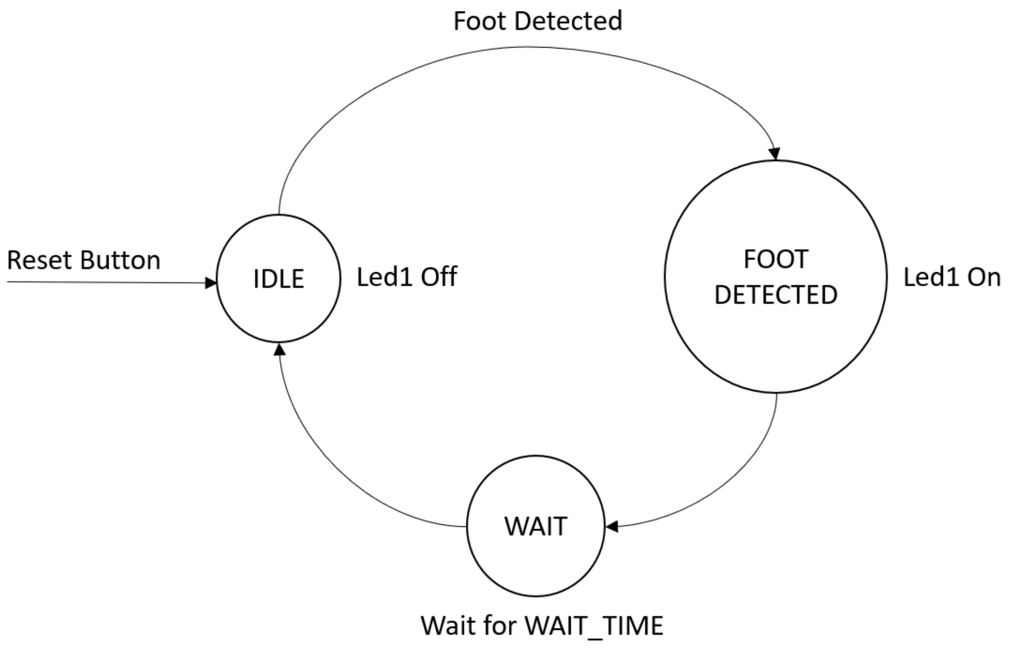


Figure 4.4: *Foot Detection state machine*

4.3 Trunk Lock and Lift

The *Trunk Lock* and *Trunk Lift* subsystems are also essential to build the *Trunk Control System*. The first allows locking and unlocking the trunk, the second allows the trunk to lift.

4.3.1 Hardware Configuration

Figure 1.4 shows the hardware configuration used to build those subsystems. Three motors are required, one for locking and two for lifting. To control all the motors, the AEK-MOT-2DC40Y1 is employed.

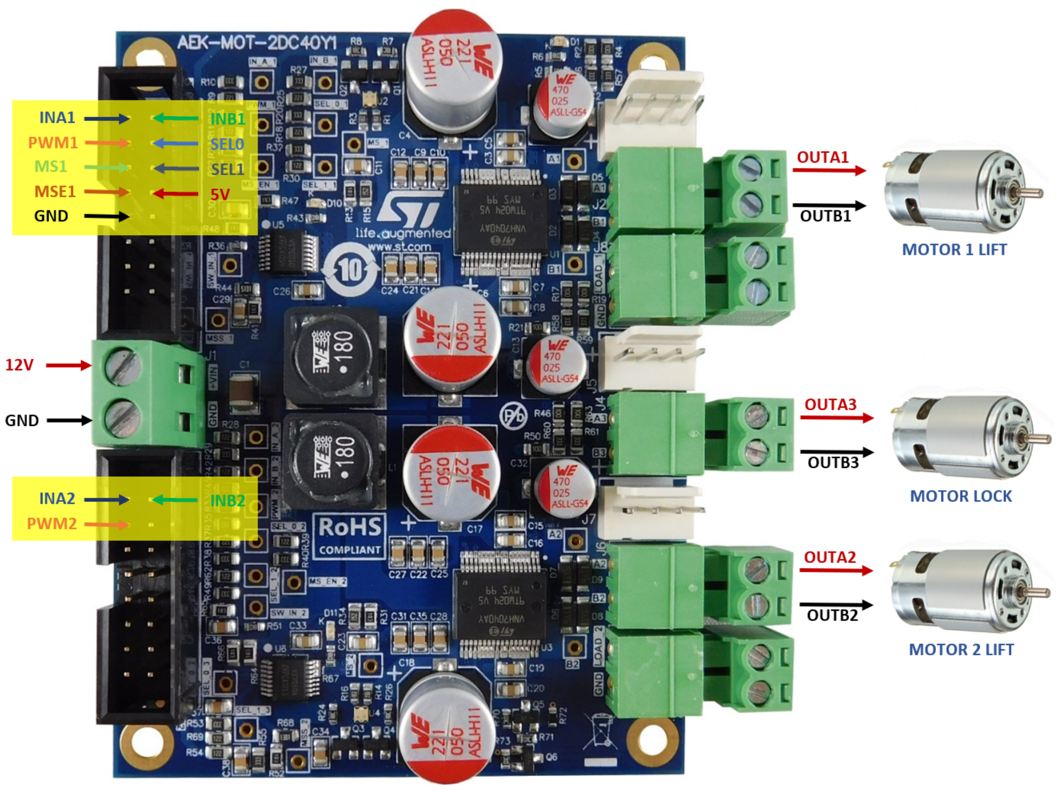


Figure 4.5: *Trunk Lock and Lift connections*

This board, as seen in Chapter 2, allows the motors to rotate in both directions thanks to the H bridge topologies embedded. Another important feature consists in sensing the current absorbed by the motors. This is really useful as it allows us to understand if the engine is subject to stress, related to the possibility that the Trunk encounters an obstacle during the opening phase. Concerning the pins, those needed to let the motors rotate are INA and INB. If INA is high and INB is low, the motor rotates in clockwise direction. In the opposite case the motor rotate in counterclockwise direction. Finally, if both signals are high the motor is in brake condition. A PWM signal can be use to modulate the motor speed rotation. All other pins are employed in the multisense feature for Motor 1.

4.3.2 Software Configuration

The system behaves as follows. In IDLE state the system waits until the *Button Open* is pressed. Than it starts the unlock procedure, starting the MOTOR LOCK engine in counterclockwise direction. After a selected time, it starts the opening procedure. The two LIFT MOTORs start to rotate in clockwise direction untile the first motor absorbs a current exceeding the selected threshold. Then the two motors are braked until the *Button Close* is pressed. Pressing this last button starts the closing procedure, mirroring the first.

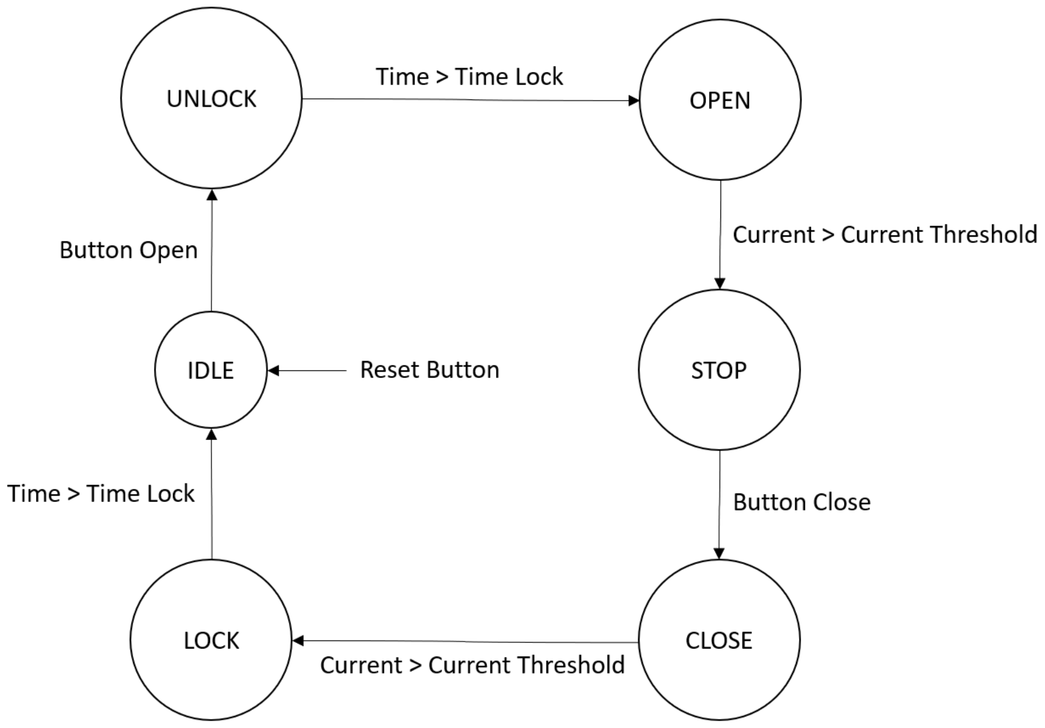


Figure 4.6: *Trunk Lock state machine*

4.4 Visual Alert

The *Visual Alert* is provided for safety reasons. Through this system the user is warned through a light blinking routine that the trunk is opening up, avoiding unpleasant accidents. The flashing of the lights must occur both in the opening and in the closing phase.

4.4.1 Hardware Configuration

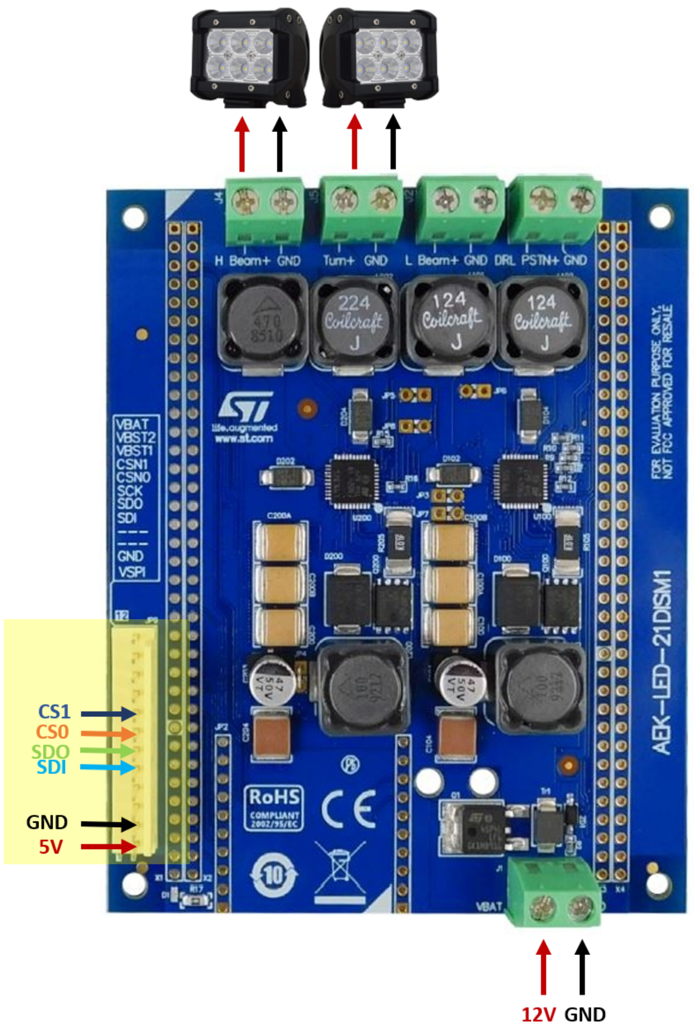


Figure 4.7: *Visual Alert board connections*

To manage the lights the AEK-LED-21DISM1 board is used. The connections are showed in Figure 4.6. The two L99LD21 led driver ICs hosted on the board require a supply of 5V. They communicate with the microcontroller through SPI protocol. All the SPI pins but the chip selects are shared among the two ICs.

4.4.2 Software Configuration

The initial state of the system is the IDLE one and it remains in this state until the *Button* is pressed. On Button press the routine for the visual signaling starts. This routine is based on cyclically switching the lights on and off so that the time the lights are on is the same as the time the lights are off. The routine ends after *N* cycles and the system return to the initial conditions.

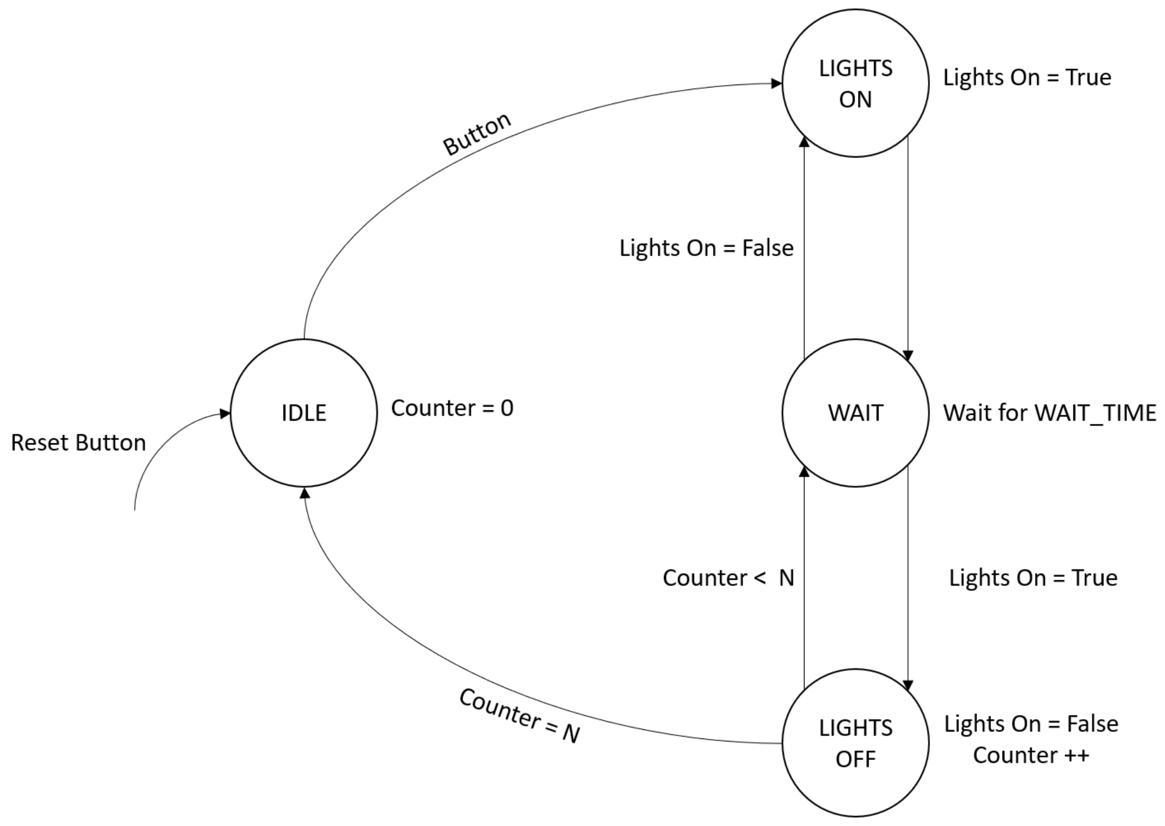


Figure 4.8: *Light Control state machine*

4.5 Acoustic Alert

The *Acoustic Alert* subsystem plays a role similar to that of the previous system. It warns with an acoustic signal that the trunk is raising or lowering. This acoustic signal is a beep synchronized with the flashing of the lights of the *Visual Alert*.

4.5.1 Hardware Configuration

This system, unlike those seen so far, involves the use of two ECUs. In addition to the AEK-MCU-C4MLIT1 master, there is a slave microcontroller stored in the AEK-AUD-C1D9031. The hardware configuration is showed in Figure 2.9. First of all the AEK-AUD-C1D9031 is powered at 12V. Two speakers are connected to the board outputs, represented in the right of the figure. The board shall be controlled via CAN bus from a master ECU. The CAN protocol requires the connection of the two pins at the top, carrying the CAN High and CAN Low signals.

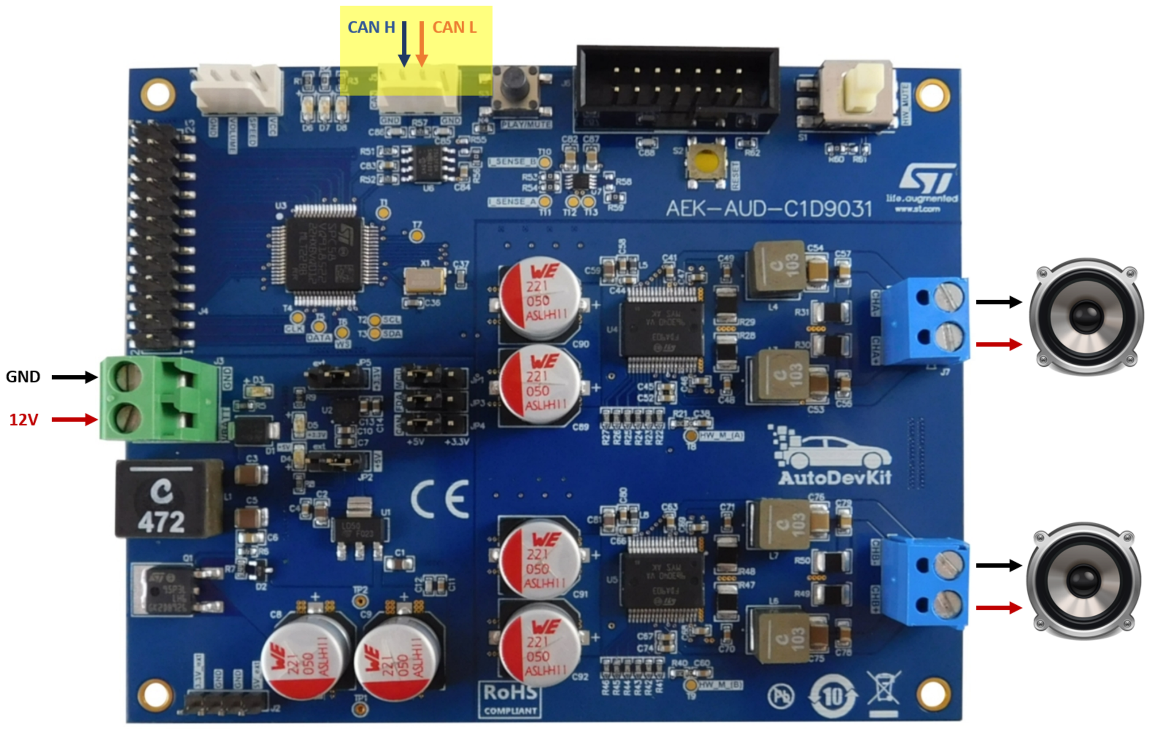


Figure 4.9: *Acoustic Alert board connections*

4.5.2 Software Configuration

Since the *Acoustic Alert* subsystem requires two ECUs, there are two firmware running that communicate with each other. The master, as showed in Figure 2.10, waits the pressure of the PLAY or STOP button to send the corresponding CAN message to the slave. On the other hand, the slave waits to receive a CAN START or STOP message, to actually play or stop the sound file stored in its flash memory.

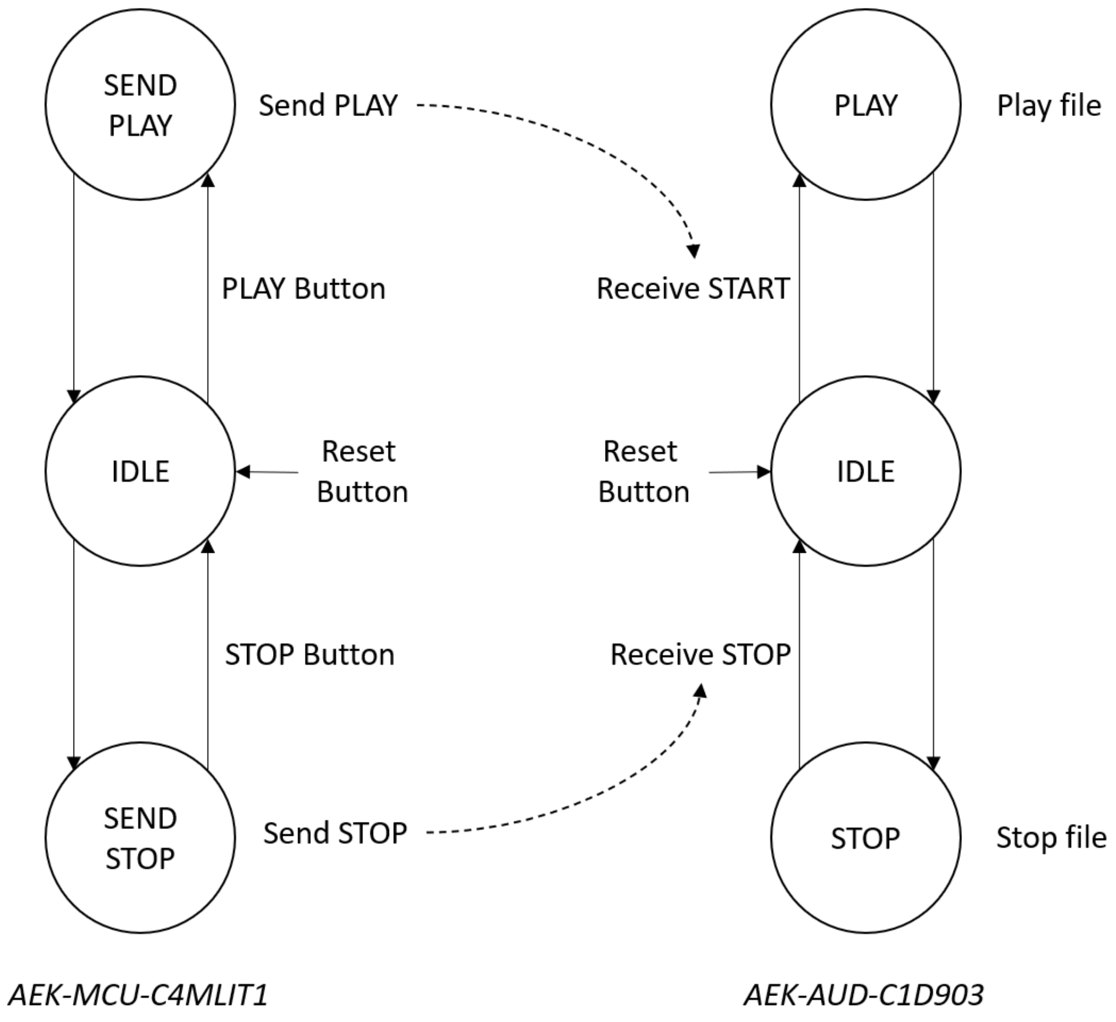


Figure 4.10: *Acoustic Alert state machines*

The file contains a sequence of beeps and has the following properties:

- **Extension**: wav
- **Bit depth**: 16
- **Channels**: 1
- **Sample rate**: 44.1KHz

4.6 Motion Control

The *Motion Control* subsystem recognizes if the car is in motion or not. If the car is in motion the trunk shall not be opened even if a foot is detected, once more for safety reasons.

4.6.1 Hardware Configuration

For the *Motion Control* the STEVAL-MKI207V1 board is used, hosting the ASM330LHH Gyroscope MEMS. This board is plugged into the AEK-CON-SENSOR1 board through the P1 connector, as showed if Figure 2.12. To connect the sensor to the supply a jumper is inserted between the left and central pins on VDD-SEL. Both VDD and VDDIO must be connected to 3.3V. Finally four pins are dedicated to the SPI Communication.

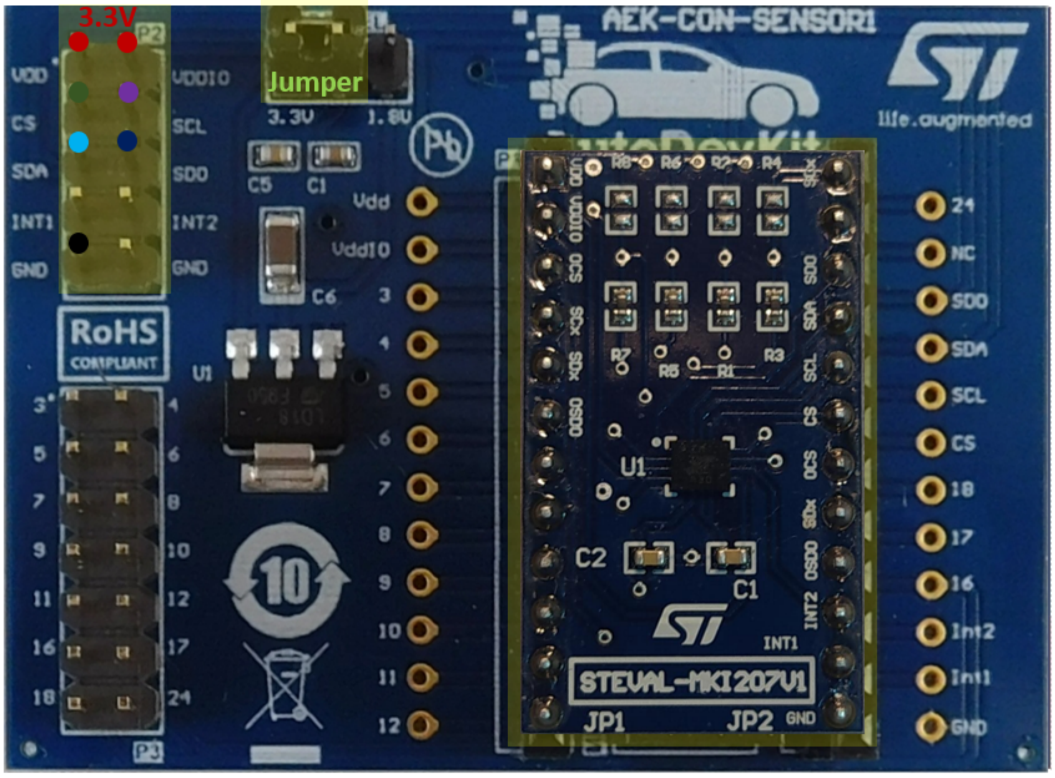


Figure 4.11: *Motion Control board connections*

4.6.2 Software Configuration

To detect if the car is in motion, the wake-up feature of the gyroscope is used. To detect this event two parameters must be configured: Threshold and Duration. If the sensor produces acceleration samples that exceed the selected Threshold for a Duration interval time, it detects a wake-up event.

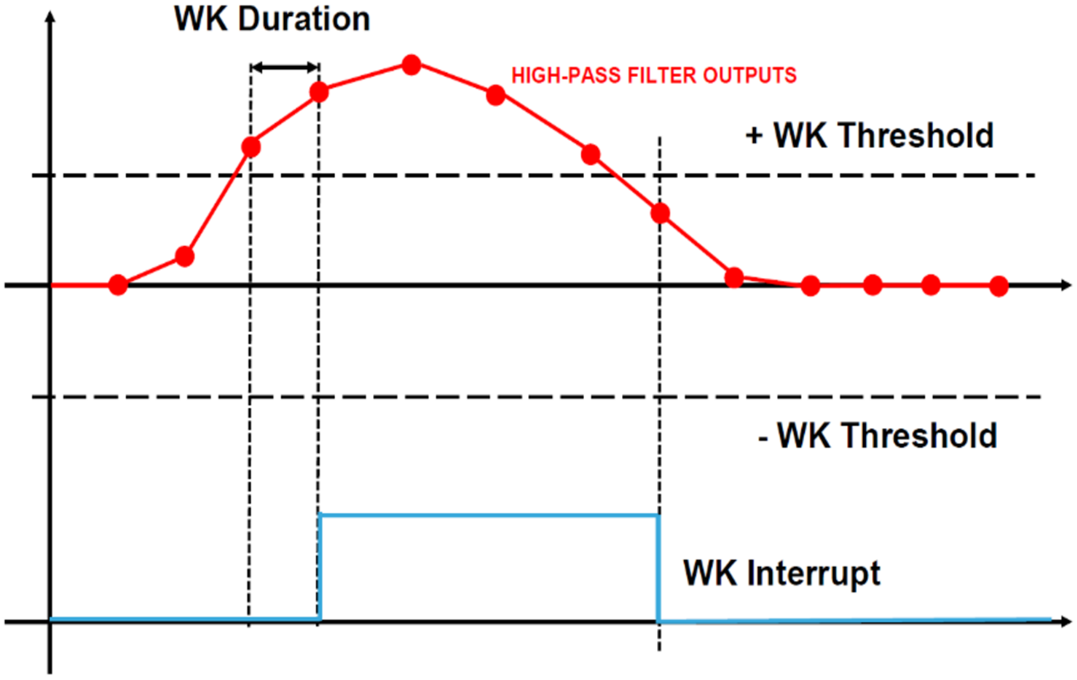


Figure 4.12: *Wake-Up Detection signals*

In this subsystem the *LED* is turned on for WAIT_TIME seconds if a motion is detected. Then it is turned off and the subsystem returns in IDLE state.

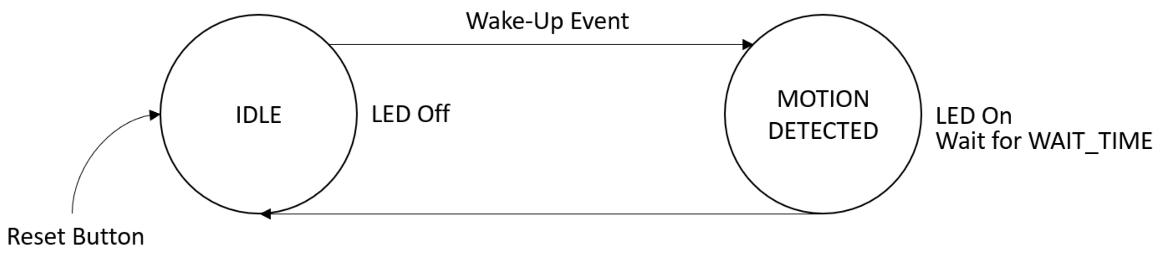


Figure 4.13: *Motion Control state machine*

4.7 Infotainment GUI

The *Infotainment* GUI shows the info related to the status of the *Trunk Control System*. It is useful because using a Display the user has a feedback of what's happening to the *Trunk Control System*. It is also useful to give input to the system through the display embedded touch panel.

4.7.1 Hardware Configuration

Both Display and Touch communicate through the SPI protocol. For this reason, as can be seen in Figure 4.14, 8 pins are employed in this communication. MOSI, MISO and CLK lines can be shared while CS and TCS must be different. The display needs 5V of power supply, and 3.3V are needed for the LED pin. In this case the IRQ pin is essential because every time the touch is pressed an interrupt is generated, allowing to detect the event.

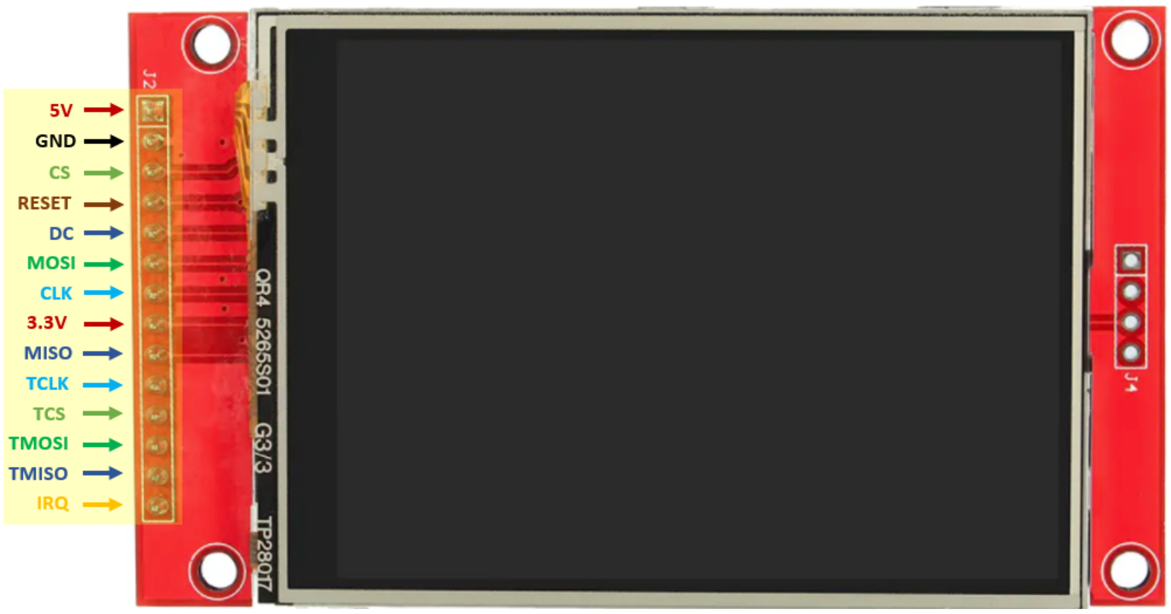


Figure 4.14: *Infotainment GUI board connections*

4.7.2 Software Configuration

The first version of the *Infotainment* implements a simple state machine. At the beginning the system is in IDLE state and the LCD is configured. Then the state is changed in PRINT MESSAGE and the first of N messages is showed through the display. Every time the touch is pressed the message cyclically changes. This state machine will be appropriately modified for the final *Trunk Control System*.

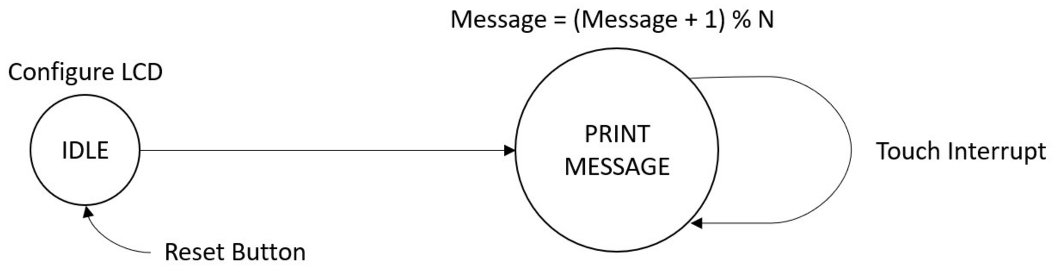


Figure 4.15: *Infotainment GUI state machine*

Chapter 5

Advanced Gesture for Foot Detection

5.1 Introduction

As said before, the *Foot Detection* subsystem is the heart of the entire *Trunk Control System*. It shall have the following features:

1. **Precise** and **Accurate** in the foot recognition

2. **Intuitive** to use

3. **Resistant** to different weather and soil conditions.

It now becomes clear that the simple *Foot Detection* presented in Chapter 4 does not meet these requirements as each measurement below the selected threshold can be mistaken for a foot and the different atmospheric and soil conditions impacts the acquisition of accurate samples by the ToF sensor.

5.2 The gesture recognition

To achieve the three key requirements, we propose a **gesture** recognition. Rather than trying to recognize a foot, we shall try to identify a sequence that must be performed by the user to get the trunk open. The choice falls in a gesture simple to reproduce by the user and at the same time difficult to reproduce by natural events, so that the first two requirements are met. The gesture to be performed is represented in Figure 5.1.

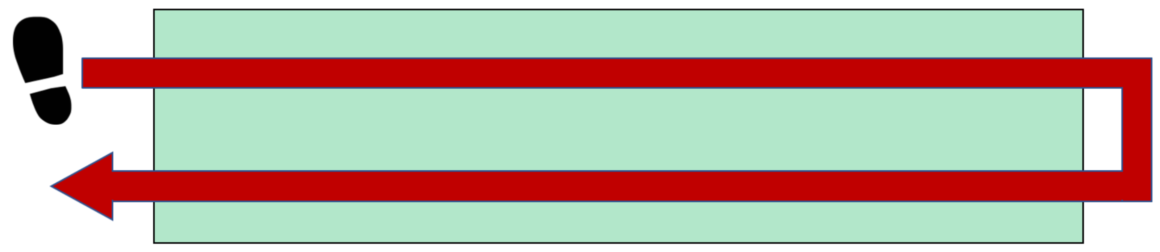


Figure 5.1: *Foot Detection gesture*

Selected a *Detecting Base*, identified by one or more sensors, the user must cross it completely in both directions by performing a cyclic movement in a fixed time lapse.

5.3 Detecting Area

The definition of the **Detecting Area** is essential to understand how high and how wide the movement to perform should be. Suppose to build a **Detecting Device**, containing one or more ToF sensors, facing downwards and projecting a *Detective Area*, as shown in Figure 5.2.

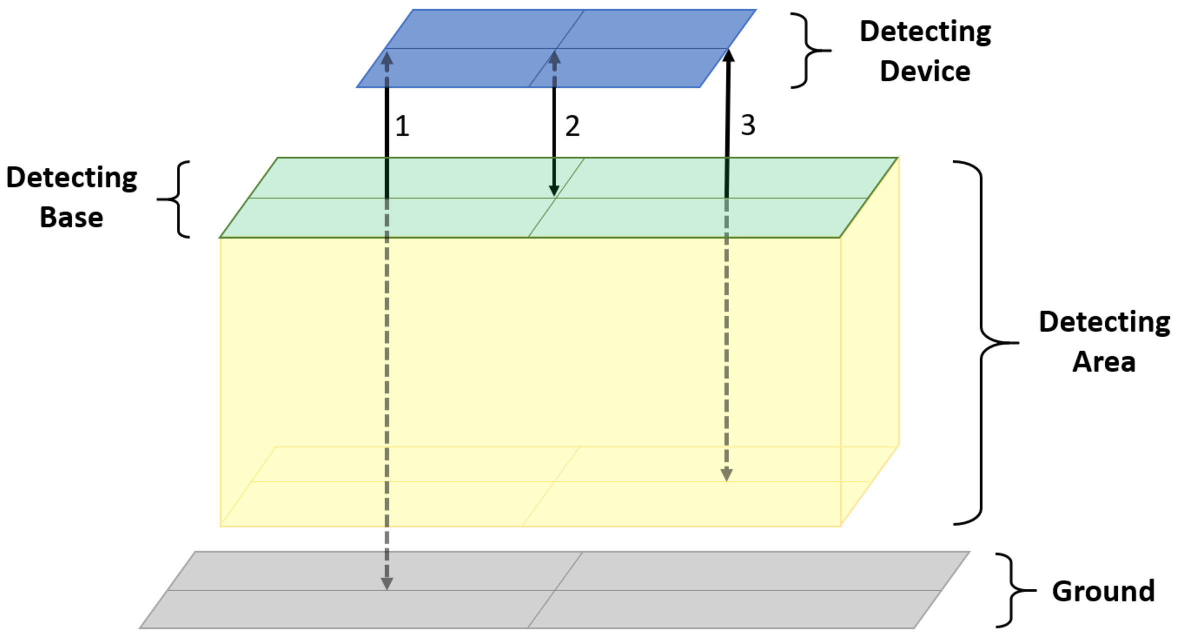


Figure 5.2: *Detection constraints*

The elements defining it are the **Detecting Base** and **distances 1, 2** and **3**. To build the gesture-based foot recognition system, these four elements must be defined. The chosen parameters are:

- **Distance 1**: distance between the *Detecting Device* and the ground. A plausible value is 30 cm.

- **Distance 2**: minimum detection distance. It shall be 0.
- **Distance 3**: maximum detection distance. It shall be as close as possible to distance 1, allowing to detect even gestures performed low to the ground.
- **Detecting Base**: the minimum surface that must be horizontally completely crossed to perform the gesture. The width shall be 20 cm and the height between 2 and 5 cm.

The next step consists in understanding if the ToF sensor is able to work at these distances.

5.4 Single Sensor Setup

The following setup is used for the single sensor studies. The ToF is placed at 30 cm from the ground, facing down, respecting the chosen *distance 1*. A 16x16 *Region of Interest* is selected because we want the sensor to cover the largest possible area assuming that the *Detecting Base* must be entirely covered by the sensor.

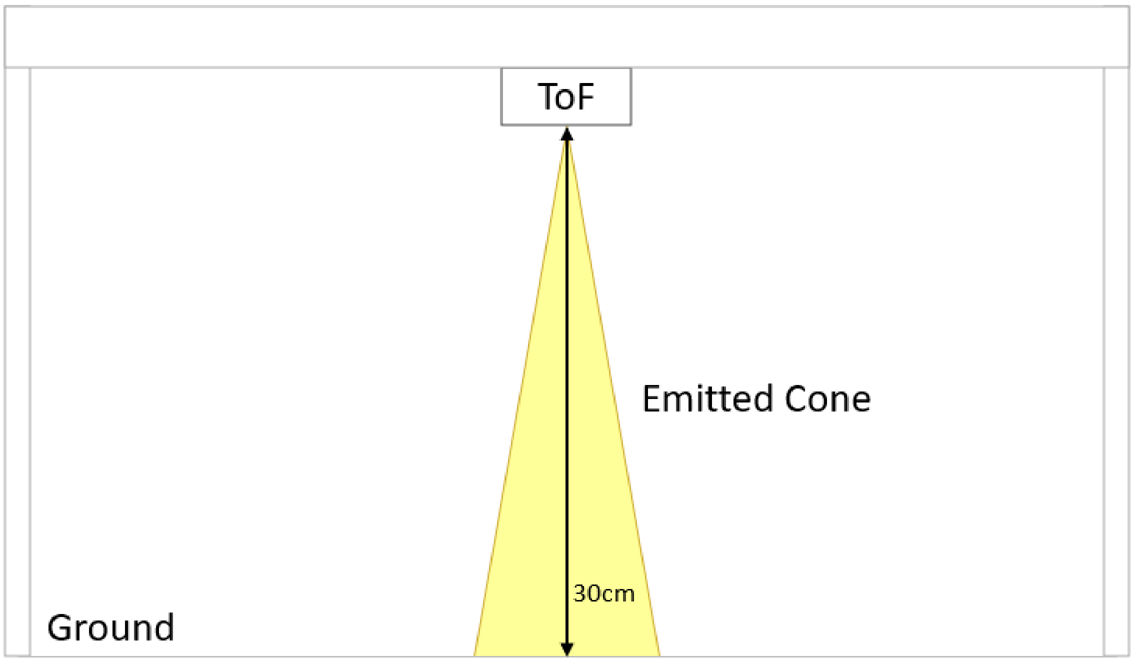


Figure 5.3: *Single sensor setup*

5.5 Time of Flight Sensor

As described in Chapter 2, the VL53L1X sensor sends an optical signal to the object through a cone of 27° and detects its distance through the reflected beam captured by the SPAD Array. The first beam reflected hitting an array element determines the acquisition of a single distance measurement. This means that for each beam sent the sensor detects just one distance. This array can ranges from a minimum dimension of 4x4 to a max of 16x16.
Regarding the constraints on the *Detecting Area*:

- **Distance 1**: is respected by construction
- **Distance 2**: can be 0 since the sensor has no restrictions on the minimum detection distance
- **Distance 3**: must be identified since the sensor is subject to disturbances related to the atmospheric and soil conditions in which it is located
- **Detecting Base**: this also requires other studies as it depends on the sensor selected ROI and *distance 1*.

5.6 Maximum Detecting Distance

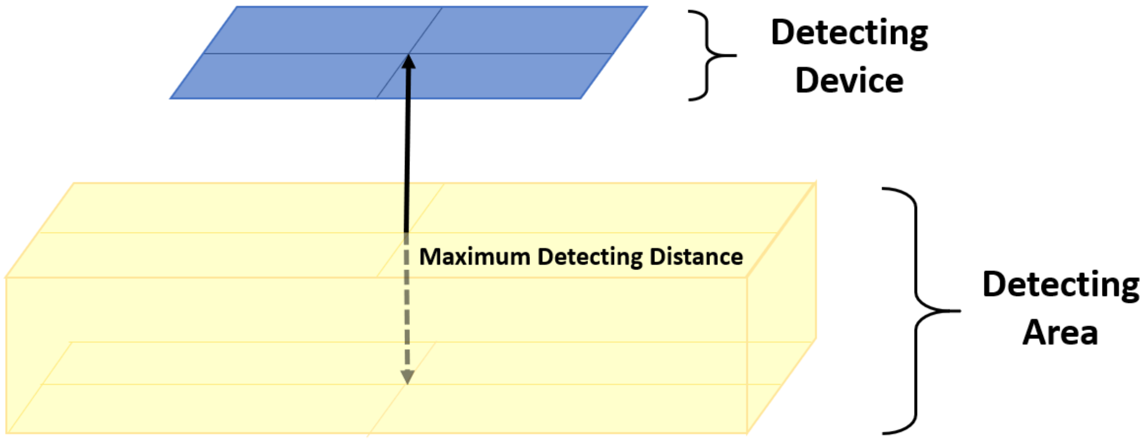


Figure 5.4: *Maximum Detecting Distance*

The *Distance 3*, referred as *maximum detecting distance* from now on, defines a threshold for the object detection so that every object situated at a distance greater than the maximum is ignored. More this distance is similar to *distance 1*, more the system works well allowing to detect movements close to the ground. In an ideal case, assuming the sensor is not making mistakes in detecting distances, the *maximum* *detecting distance* and *distance 1* could be the same, so that every object above the ground can be detected.

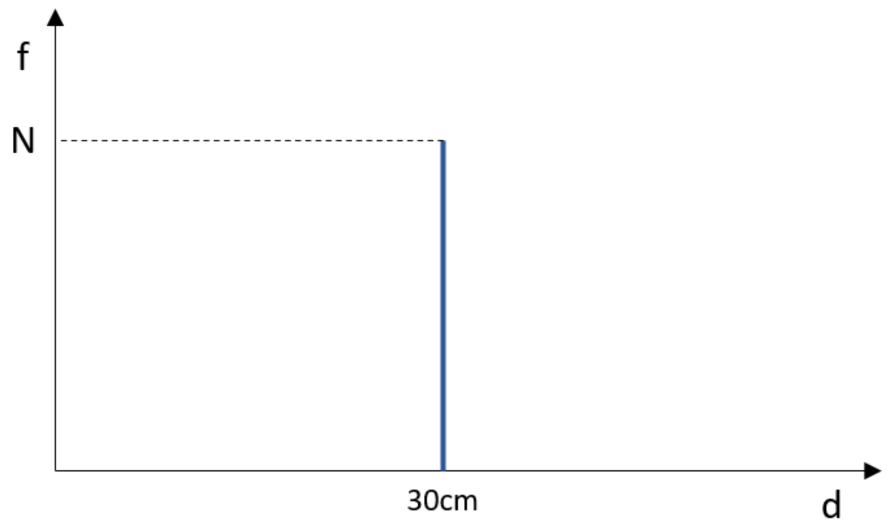


Figure 5.5: *Ideal Detection. N measurements of the ground.*

Unfortunately the sensor does not always detect exactly the same distance but shows a behavior like the one presented in Figure 5.6.

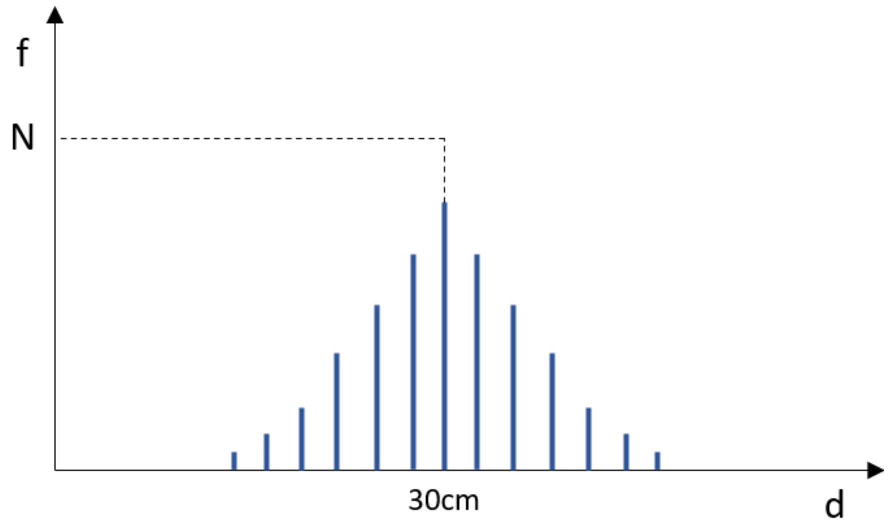


Figure 5.6: *Real Detection. N measurements detect a distribution of values.*

It happens because the sensor works with light rays and the distance measurement depends on their reflections. To determine the *maximum detecting distance* it is necessary to quantify the error in the sensor measurement under different weather conditions. For this reason, several tests were carried out producing the following results.

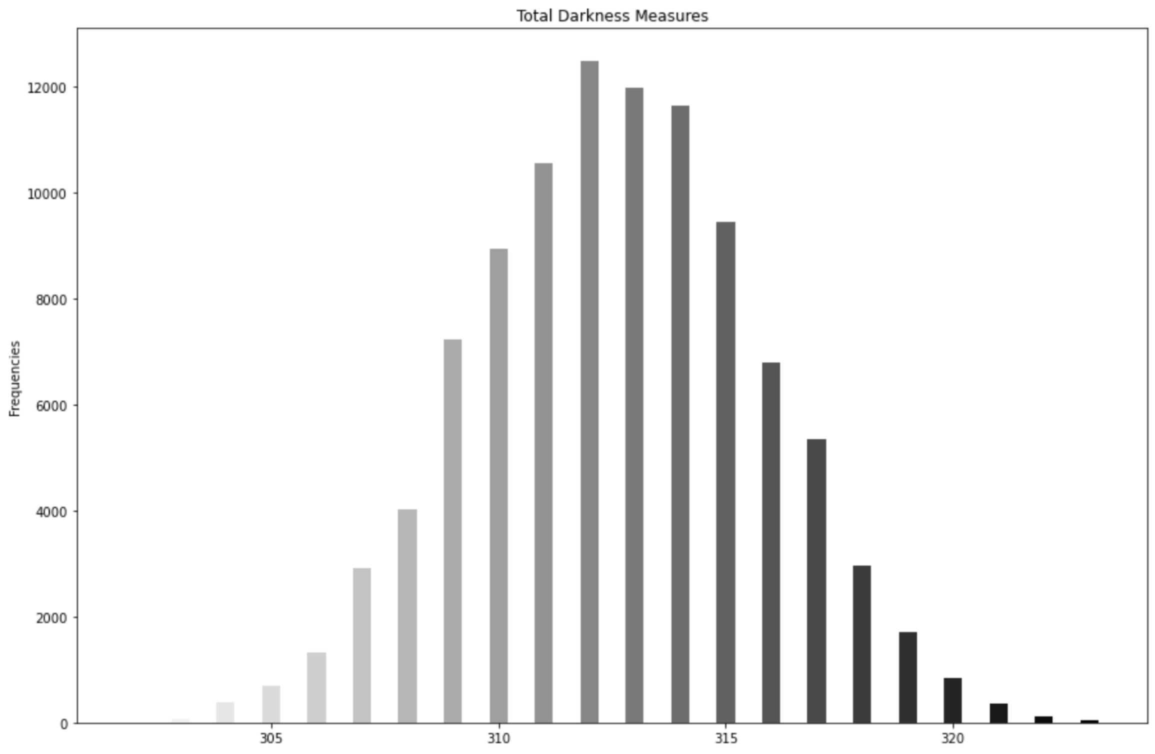


Figure 5.7: *Total darkness measures*

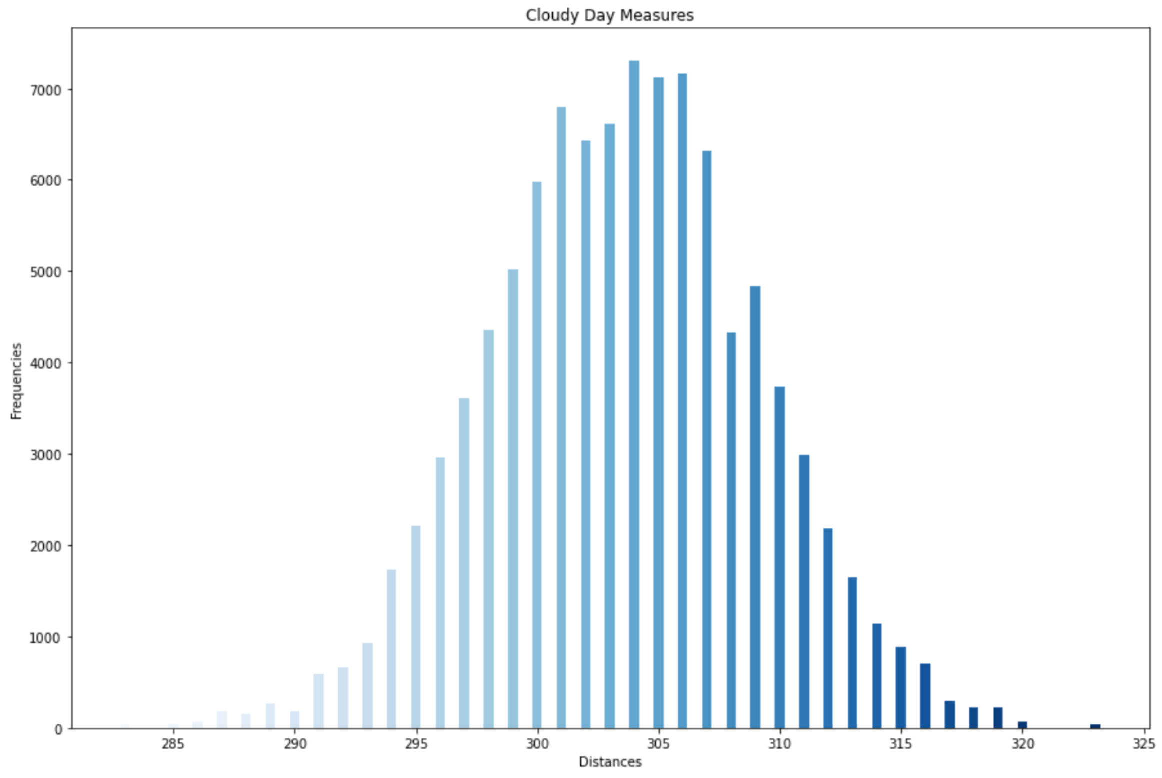


Figure 5.8: *Cloudy day measures*

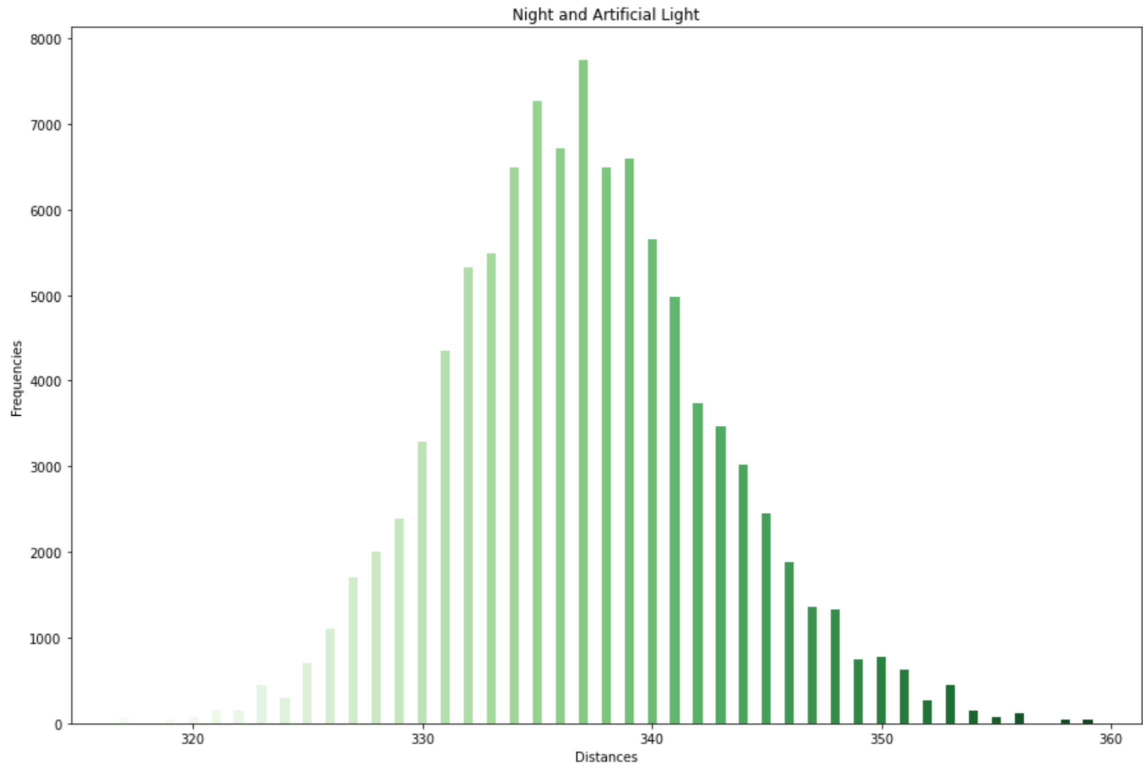


Figure 5.9: *Night and artificial light measures*

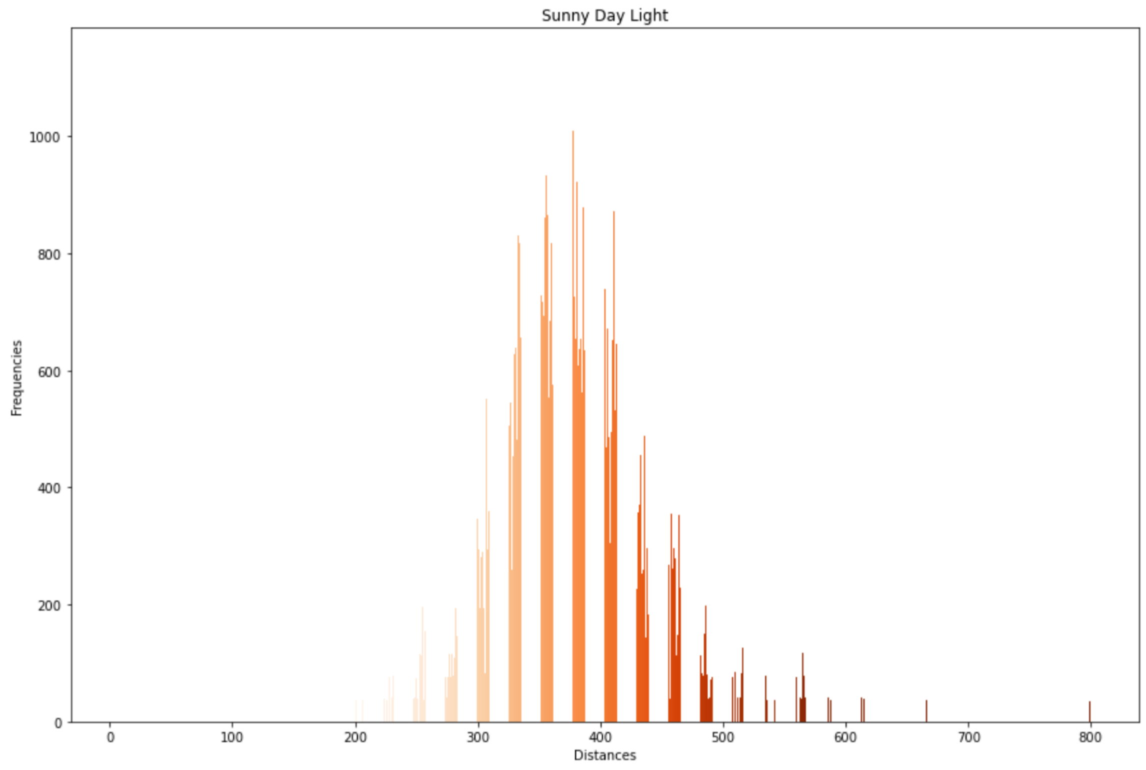


Figure 5.10: *Sunny day measures*

To get the data distributions, a Python script was developed, including PySerial and Matplotlib packages. The first allowed to collect data via serial, the second to plot them. For each test 1x10^5 samples were analyzed.
From data distribution, it can be noticed that the environment in which the sensor is placed determines different measurements. Less is the environment light (both artificial and natural) less becomes the range of values detected. This is proven by the distribution of Figure 5.7, detected in total darkness condition. The worst measurements belongs to the sunny day case where the average value is 10 cm higher than the real one and the range of values goes from 20 to 80cm.
To minimize the probability of error in detecting an object when there is no one, the minimum distance value between the various distributions should be considered. So, the *Maximum Detecting Distance* chosen is 20 cm.

5.7 Detecting Base: Single Sensor

The *Detecting Base* determines the surface that the foot must completely cross for the gesture recognition. In particular the *width* must be completely crossed as the gesture develops horizontally. However the *height* is also important because the wider it is the easier it is to be intercepted by a foot.

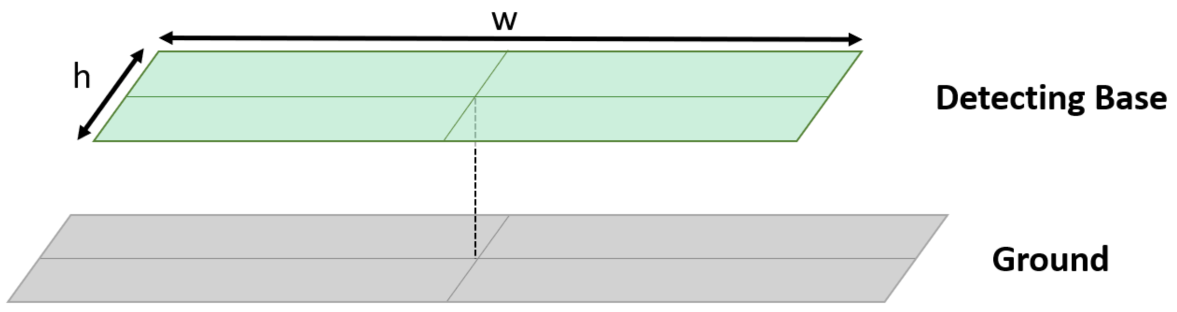


Figure 5.11: *Detecting Base*

These parameters depends on the selected *Region of Interest*. A bigger ROI covers a bigger surface but the measurements obtained by its edges are less accurate. Therefore, the decision is to consider the *Detecting Base* as the surface covered by the sensor which provides an error on the measurements less than 15%, thus excluding measurement errors greater than 4.5 cm. This means that an error greater than 4.5 cm can not be tolerated. A test is performed to know the maximum *w* and *h* of the *Detecting Base* that can be afforded by the sensor. A cube is placed on the ground, centered with respect to the sensor, and it is translated along the *w* axis to detect its distance. Note that because a 16x16 SPAD is selected, *h* and *w* are equals.

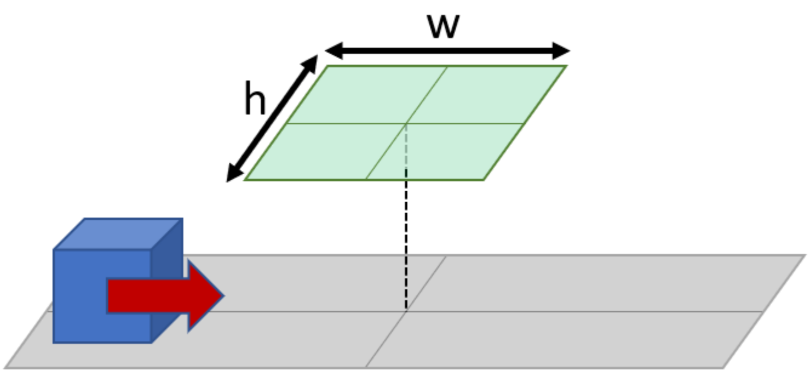


Figure 5.12: *Cube placement*

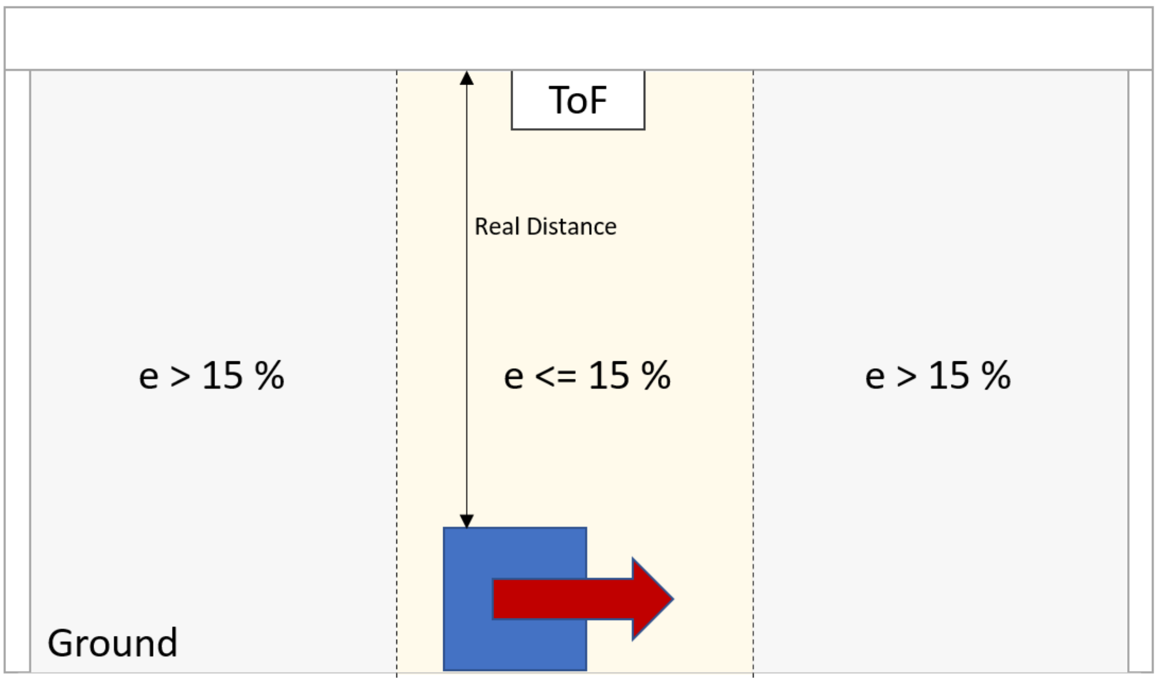


Figure 5.13: *Test to find the Detecting Base*

The errors are computed as:

$$e = Real\ Distance - Measured\ Distance \tag{5.1}$$

The test shows that the *Maximum Detecting Base* achievable with one sensor is a square with side 4 cm. That is not enough according to the system requirements so at least two sensors are required.

5.8 Detecting Base: Double Sensor

We assumed that the *Detecting Base* is the minimum surface to be crossed while performing the gesture. It does not necessarily means that both sensors must cover the entire area. It is possible to place two sensors at the edge of the area defining the *Detection Base* itself, showed in Figure 5.14.

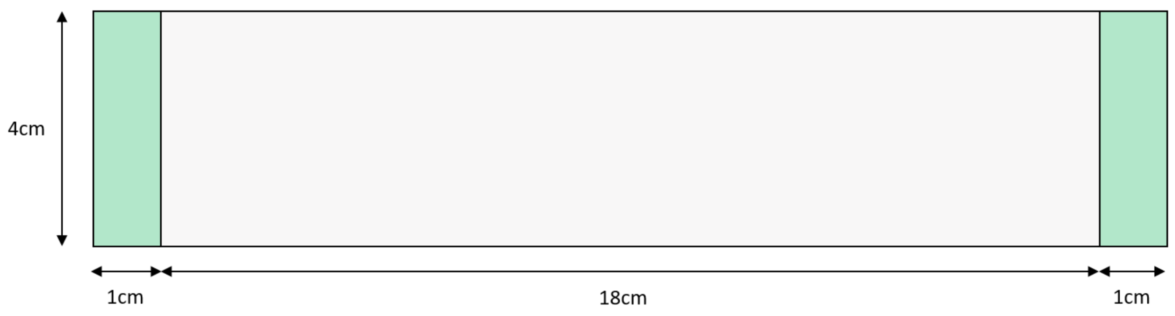


Figure 5.14: *Detecting Base with two sensors*

This area is obtained selecting a 4x16 ROI and placing the sensors at 19cm of distance, as showed in Figure 5.15.

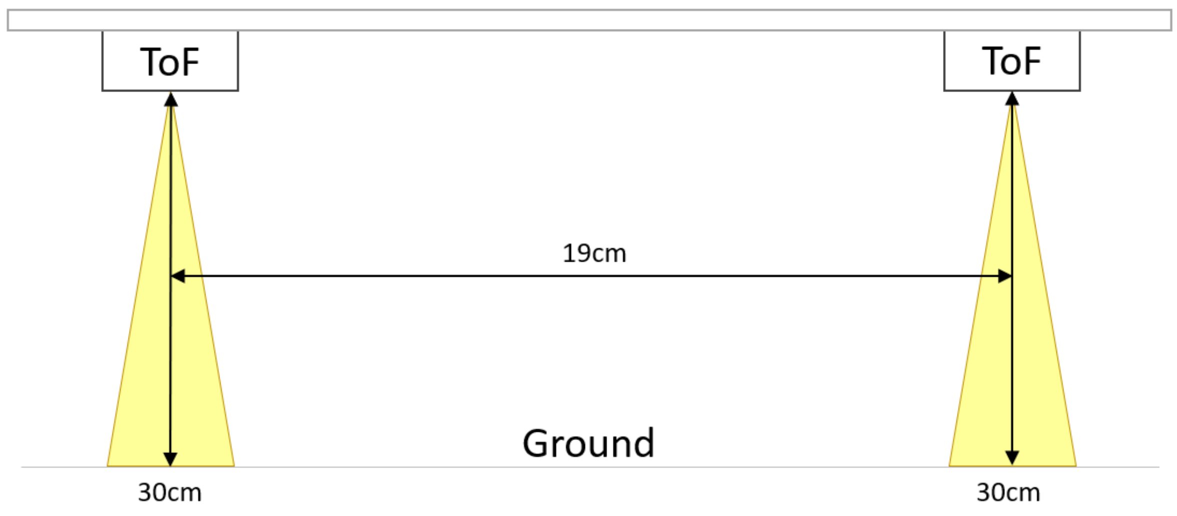


Figure 5.15: *Two sensors hardware setup*

The decision to minimize the width of the area covered by each sensor (green areas) maximizing instead the one not covered (grey) was taken to make the system more robust. This makes it harder for one foot to cover both areas without proper recognition of the gesture.

5.9 The Gesture Recognition

As said before, the *Trunk Opening Control System* starts only if the gesture performed by the foot is recognized. The gesture consists in crossing in both directions the *Detecting Base*, as showed in Figure 5.1.

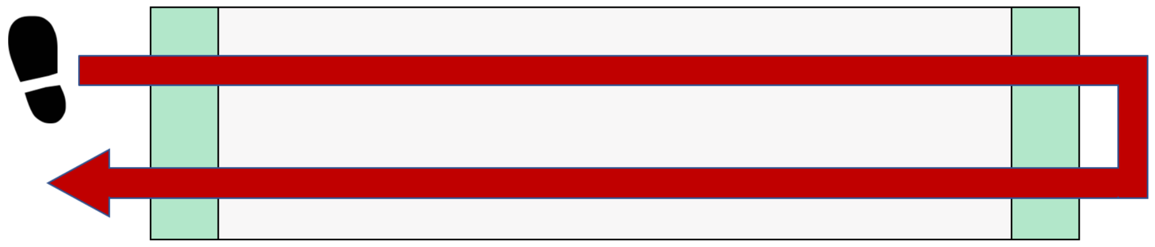


Figure 5.16: *Gesture on the new Detecting Base*

The algorithm is based on the idea that a foot can be in three different **states**:

- **0**: not detected.
- **1**: detected by sensor 1.
- **2**: detected by sensor 2.

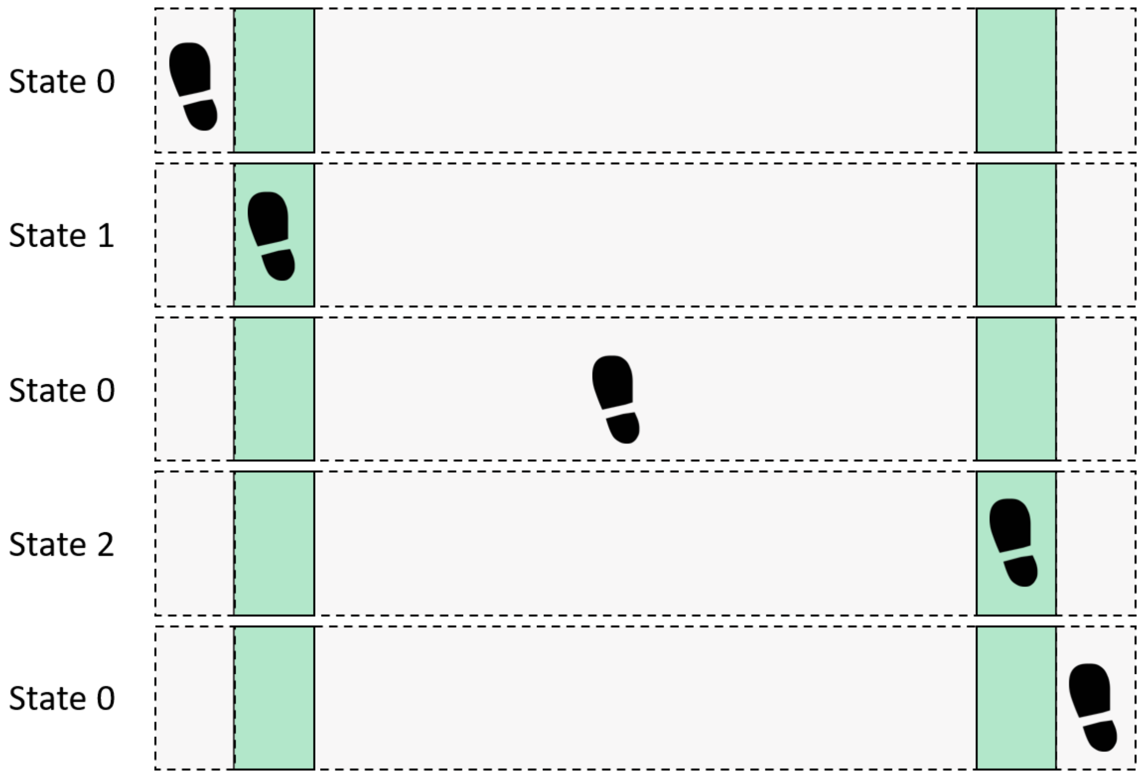


Figure 5.17: *Foot Detection System states*

A gesture can be represented through a sequence of *states*. The sequence that represents the gesture of Figure 5.16 is: **0,1,0,2,0,2,0,1,0**. Also the sequence **0,2,0,1,0,1,0,2,0** is admitted because we want the gesture to work also in the opposite direction. The central zeros in the sequence are necessary because a complete crossing of the detecting base is desired. For example the first admitted sequence is showed in Figure 5.18.

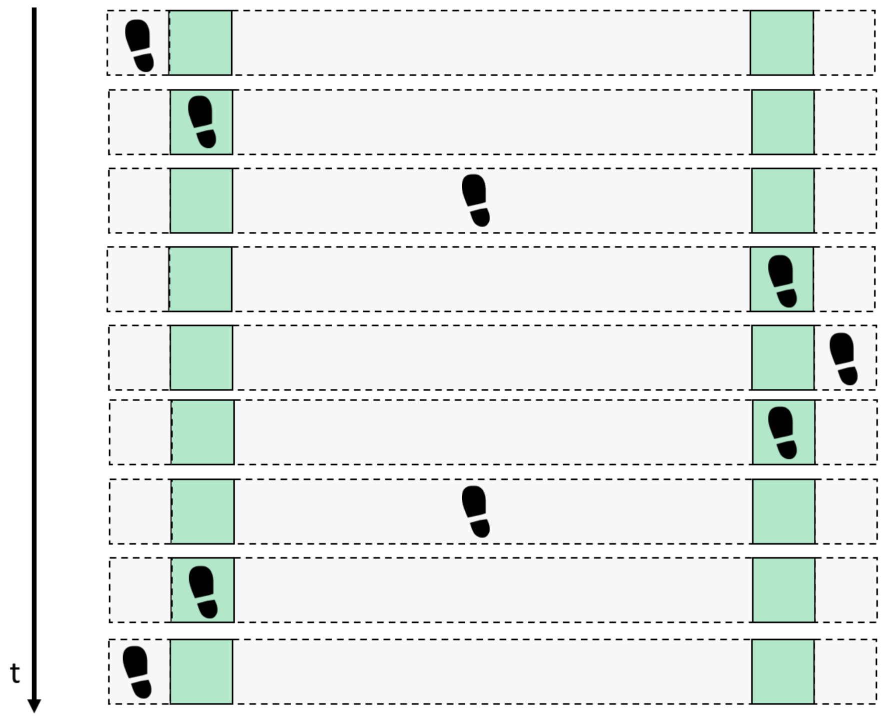


Figure 5.18: *Accepted sequence*

The algorithm is implemented using the function *detect_foot()*, called cyclically by the microcontroller. It can be divided in four parts:

- **System State Detect**: detect the *state* of the system. If *sensor 1* detects a distance less than the *maximum detecting distance* the *state* is *1*; if *sensor 2* detects an object the *state* is *2*; if there is no detection the *state* is *0* otherwise, if both sensors detect something, the *state* is *3*. The last means that there is a fixed object under the trunk, e.g. a stone.
- **State Sequence Update**: update the *sequence* of *states* if the *state* changes.
- **State Sequence Check**: compares the detected *sequence* with those admitted, if the first reaches a length equal to the SEQ_LEN. If there is a match the function set the global variable *foot_detected* to 1
- **System Reset**: if the sequence reaches a length equal to the SEQ_LEN or a *RESET_TIME* has elapsed, the system resets and it re-starts the detection mode.

```c
void detect_foot(){
  static uint16_t distance;
  static uint8_t system_state;
  static uint8_t sequence[SEQ_LEN] = {0,0,0,0,0,0,0,0};
  static uint8_t seq_index = 0;
  static uint8_t sequence_1[SEQ_LEN] = {1,0,2,0,2,0,1,0};
  static uint8_t sequence_2[SEQ_LEN] = {2,0,1,0,1,0,2,0};

  /* Detect System State */
  system_state = 0;
  get_distance(&distance, AEK_TOF_DEV0);
  if(distance < MAX_DETECTING_DISTANCE) system_state += 1;
  get_distance(&distance, AEK_TOF_DEV1);
  if(distance < MAX_DETECTING_DISTANCE) system_state += 2;

  /* Update Sequence */
  if(system_state + seq_index > 0 && sequence[seq_index-1] !=
   system_state){
    if(!seq_index) seconds = 0;
      sequence[seq_index] = system_state;
      seq_index ++;
  }
  else{}

  /* Check if Sequence is admitted */
  if(seq_index == SEQ_LEN){
    if(memcmp(sequence, sequence_1, sizeof(sequence)) == 0
      || memcmp(sequence, sequence_2, sizeof(sequence)) == 0){
        foot_detected = 1;
    }
    else {}
  }
  else{}

  /* Reset System */
  if(seq_index == SEQ_LEN || seconds > RESET_TIME){
    memset(sequence, 0, sizeof(sequence));
    seq_index = 0;
    seconds = 0;
  }
}
```

Listing 5.1: Foot Detection implementation procedure

Chapter 6

Trunk Control System

6.1 Introduction

All the subsystems realized and described in Chapters 4 and 5 are combined to build the *Trunk Control System*.

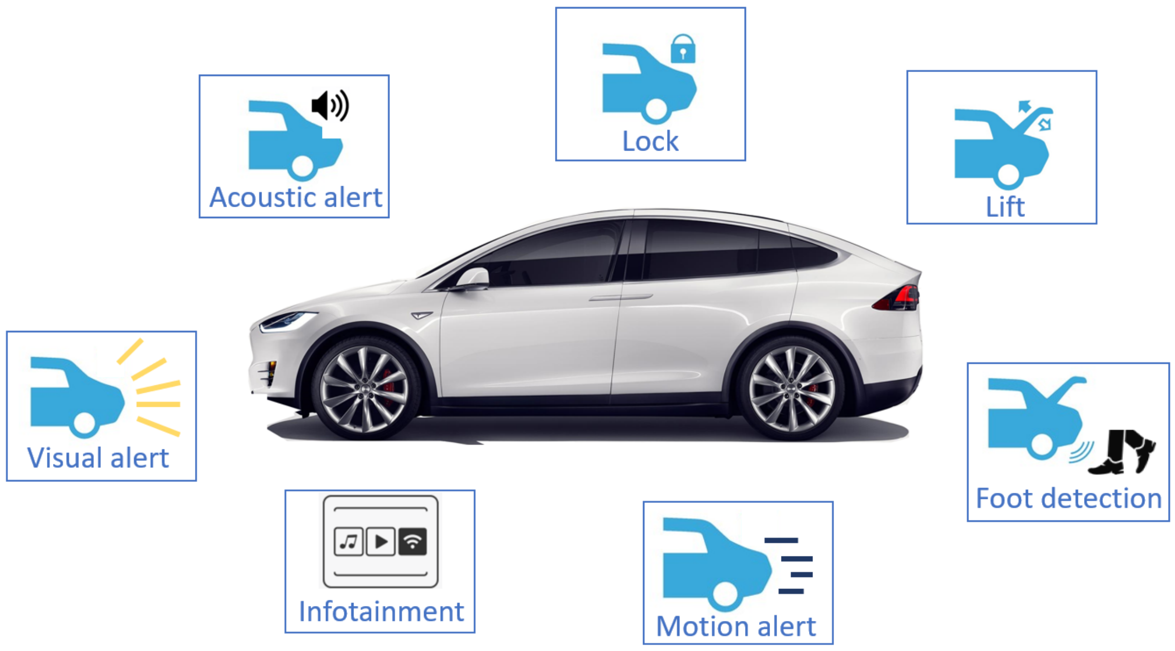


Figure 6.1: *Trunk Control System*

In summary, the *Trunk Control System*behaves as follows: it waits until the gesture is performed, unlock the trunk, raise the boot and alerts the user through audible and visual signals. In addition, it detects if the car is moving and if there is an obstacle during the opening and closing phases. A dedicated GUI displays the system status and allows user interactions.

6.2 Hardware Setup

All the boards of the *Trunk Control System* are arranged and screwed onto a plexiglass **panel** and attention is put in optimizing the length of the connection cables. The *AEK-CON-5SLOTS1* connection board is essential to arrange the cables and to secure the connections. The *panel* was designed using the *LibreCAD* Tool and made though a laser engraving machine. The CAD represented in Figure 6.2 shows the placing of the boards and the holes, needed to fix them to the *panel*.

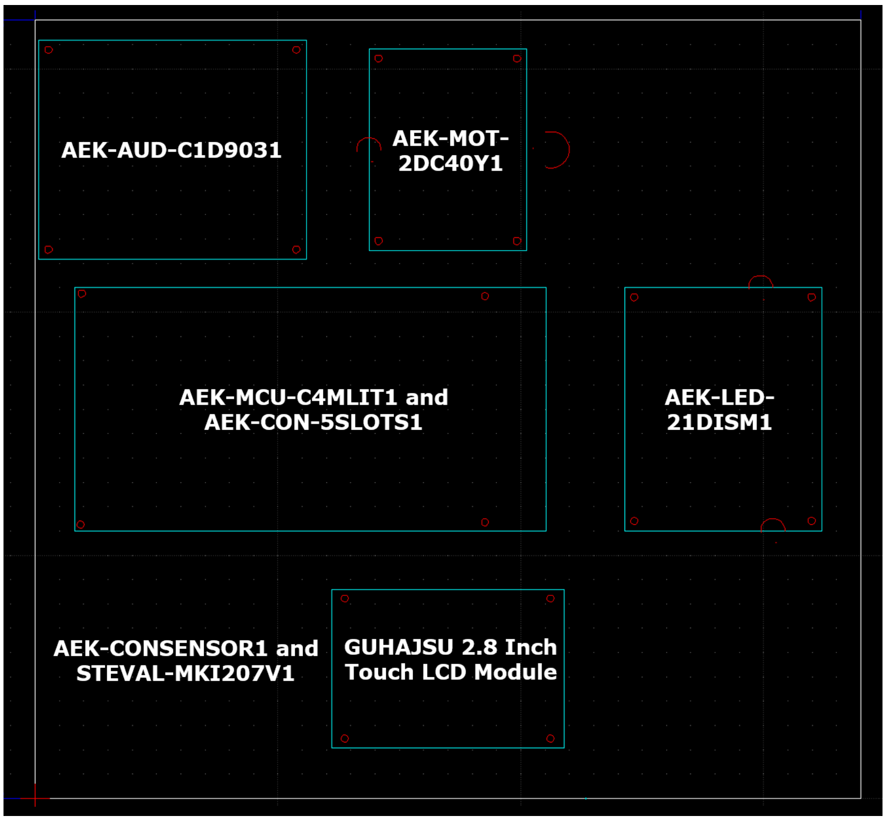

AEK-AUD-C1D9031
AEK-MOT-2DC40Y1
AEK-MCU-C4MLIT1 and AEK-CON-5SLOTS1
AEK-LED-21DISM1
AEK-CONSENSOR1 and STEVAL-MKI207V1
GUHAJSU 2.8 Inch Touch LCD Module

Figure 6.2: *Panel CAD*

The ToF sensors are placed in a dedicated **support**, designed with Fusion360 Tool and realized with a 3D printer. The *support* houses two Time of Flight sensors at 18 cm from each other, respecting the constraints imposed in Chapter 5.

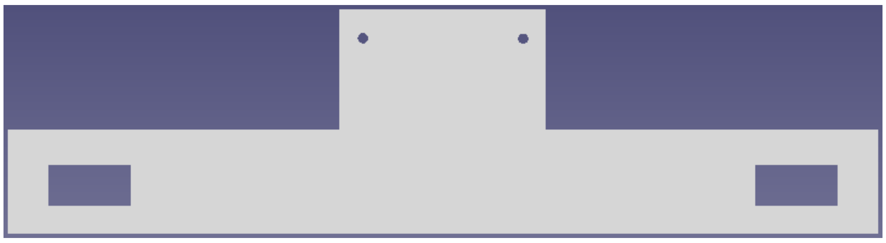


Figure 6.3: *Support for ToF sensors CAD. View from Top.*

6.3 State Machine

The State Machine of Figure 6.4 summarizes the operation of the *Trunk Control System*, based on 5 states:

- **INIT**: Initialization state. It initializes all system components.
- **WAIT OPENING INPUT**: The system waits until a gesture is detected. If that happens and the vehicle is not moving, it goes to the OPENING state.
- **OPENING**: The trunk is unlocked, the acoustic and visual alert starts and the trunk is raised.
- **WAIT CLOSING INPUT**: It stop the visual and acoustic alert. Then the system waits until the close button is pressed on the touch panel.
- **CLOSING**: The acoustic and visual signal start and the trunk is closed. If an obstacle is detected the system returns to the previous state, otherwise it moves to the WAIT OPENING INPUT state.

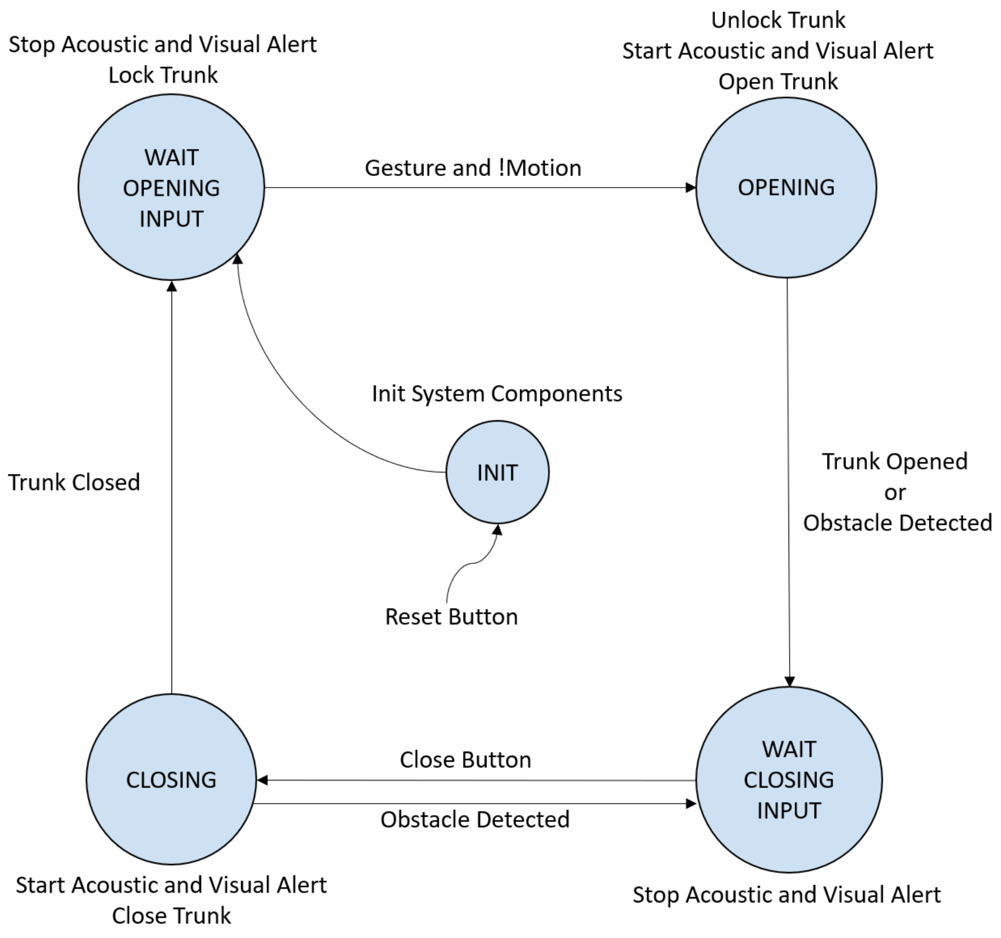


Figure 6.4: *Trunk Control state machine*

6.4 DEMO

To verify all the features of the *Trunk Control System* a DEMO was assembled. It allows the user to retrace all the states in which the system can be and to interact with it.

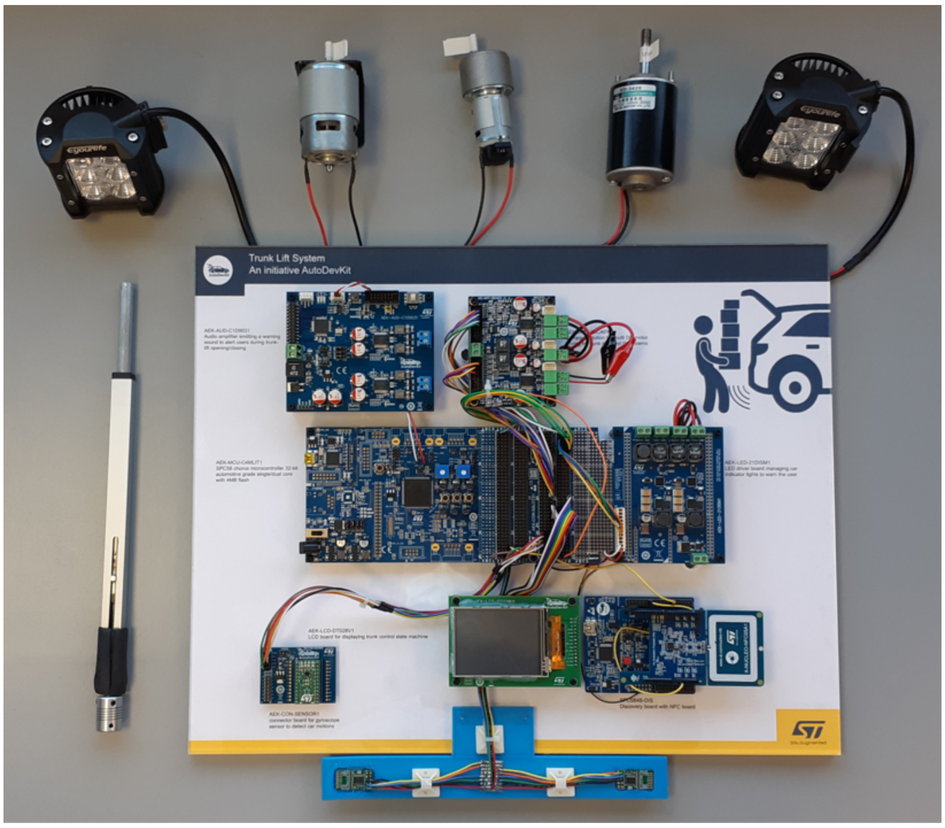


Figure 6.5: *Trunk Control DEMO.*

Note that for the DEMO realization the display is replaced with an AutoDevKit LCD board prototype and an ST NFC Reader is added, so that the opening of the Trunk can also be started approaching an NFC tag.

Chapter 7

Conclusion

In this thesis work, a *Trunk Control System* was developed using the tools belonging to the *AutoDevKit* ecosystem. The developed system is intuitive, affordable, effective but, above all, innovative. It has several features, as it can: detect a foot; unlock, lock, raise and lower a car trunk; alert the user through visual and acoustic signals during the opening and closing phases; detect if the car is in motion; understand if the trunk encounters an obstacle. The last three features also make the system safe. The innovation consists in the use of the *Time of Flight* sensors, together with an algorithm based on a gesture recognition. This kind of solution allows to use only two sensors and few computational resources, keeping the cost low. To demonstrate the potential of this system, a DEMO was created, allowing the user to analyze it up to the single detail level.

A possible development of the thesis may consist in designing a *Printed Circuit Board* integrating the *Foot Detection* subsystem with a microcontroller dedicated to the gesture recognition. Furthermore, although the gesture recognition has proved to be effective, an alternative solution may consist in developing another kind of algorithm, for example time-based, or even a machine learning model to identify the foot. Other developments may include a more accurate detection of the car motion based on multiple sensor feedback. Finally, an integration of the boards is required to fit space constraints of a typical vehicle.